AD-A221 518

[illegible]

[illegible]

MAY 1 1 1990

[illegible]

[illegible]

90 05 11 027

REPORT DOCUMENTATION PAGE

Form Approved CPM No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE Novemoer 1989	3. REPORT TYPE AND DATES COVERED Final Report

4. TITLE AND SUBTITLE

Ada 9X Project Report, A Study of Implementation-Dependent Pragmas and Attributes in Ada, November 1989

5. FUNDING NUMBERS

C = MDA-903-87D-0

6. AUTHOR(S)

Kenneth J. Fowler
John B. Goodenough, editor

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZAT REPORT NUMBER

SEI-89-SR-19

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Ada Joint Program Office
1211 South Fern St., 3E113
The Pentagon
Washington, DC 20301-3080

Ada 9X Project Office
AF Armament Lab/FXG
Eglin AFB, Florida 32542-5434

10. SPONSORING/MONITORIN REPORT NUMBER

11. SUPPLEMENTARY NOTES

This report has been produced under the sponsorship of the Ada 9X Project Office. It is one in a series that addresses special issues relevant to the Ada revision effort.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Ada Standard describes an assortment of pragmas and attributed that a compiler must support. A compiler may add other pragmas and attributes as long as the legality of the program is not affected. This study examines implementation-dependent pragmas in existing compilers with the goal of identifying candidates for inclusion in a revised version of the language. Compiler support for language-defined pragmas is first characterized and we show that the level of support varies surprisingly between implementations. A change in the Standard's Appendix F is recommended to help remedy this problem.

More than one-quarter of the implementation-defined pragmas (27%) serve to extend the functionality of pragma INTERFACE. Overall compiler uniformity would be improved by standardizing some of these extensions.

A variety of implementation-defined pragmas are provided to improve program performance. Some of the se pragmas yield well-defined results only if programs obey certain restrictions. The Standard should provide greater control over the use of these pragmas.

In contrast to pragmas, we found very few implementation-dependent attributes, and none that seemed of sufficiently wide interest to merit inclusion in a revised version of the language.

14. SUBJECT TERMS

Ada 9X, pragmas, attributes, Pragma INTERFACE, Ada Joint Program Office, Ada 9X Project Office

15. NUMBER OF PAG

82

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT UL	20. LIMITATION OF A

2

Ada 9X Project Report

A Study of Implementation-Dependent Pragmas and Attributes in Ada

November 1989

Office of the Under Secretary of Defense for Acquisition

Washington, D.C. 20301

SEI-89-SR-19
November 1989

A Study of Implementation-Dependent Pragmas and Attributes in Ada

Kenneth J. Fowler
Software Engineering Institute

This report has been produced under the sponsorship of the Ada 9X Project Office. It is one in a series that addresses special issues relevant to the Ada revision effort. John B. Goodenough, of the Software Engineering Institute, has served as the editor and coordinator for each report.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Table of Contents

List of Figures

1. Implementation-Dependent Pragmas and Attributes

Abstract: The Ada Standard describes an assortment of pragmas and attributes that a compiler must support. A compiler may add other pragmas and attributes as long as the legality of the program is not affected. This study examines implementation-dependent pragmas in existing compilers with the goal of identifying candidates for inclusion in a revised version of the language. Compiler support for language-defined pragmas is first characterized and we show that the level of support varies surprisingly between implementations. A change in the Standard's Appendix F is recommended to help remedy this problem.

More than one-quarter of the implementation-defined pragmas (27%) serve to extend the functionality of pragma INTERFACE. Overall compiler uniformity would be improved by standardizing some of these extensions.

A variety of implementation-defined pragmas are provided to improve program performance. Some of these pragmas yield well-defined results only if programs obey certain restrictions. The Standard should provide greater control over the use of these pragmas.

In contrast to pragmas, we found very few implementation-dependent attributes, and none that seemed of sufficiently wide interest to merit inclusion in a revised version of the language.

1.1. Purpose

The Ada 9X Project Office tasked the Software Engineering Institute (SEI) to conduct a study of implementation-dependent pragmas and attributes. The description of the task reads:

> *A variety of implementation-dependent pragmas and attributes have been developed by compiler vendors. The SEI shall survey all such implementation dependencies, documented in Appendix F by each vendor, to determine if some of the pragmas and attributes should be added to the Ada language standard in order to improve uniformity among compilers. The product of this task is an Implementation-Dependent Pragmas and Attributes Study Report.*

This report addresses the above task by examining the documentation provided for 143 Ada compiler implementations. The information was gathered from each compiler's Appendix F as provided in Validation Summary Reports (VSRs) for ACVC versions 1.9 and 1.10[1] and from the product reference manuals of Ada compilers available at the SEI.

[1] ACVC refers to the Ada Compiler Validation Capability. The VSR describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A [AVO 88].

1.2. Organization and Terminology

The next chapter contains the conclusions of this report. Increasingly detailed information is then offered in later chapters. The *Analysis* chapter provides a summary of the functions supported by the implementation-defined pragmas and attributes found in the sample. The *Sample Findings* chapter then provides vendor-specific detail for each pragma and attribute.

The report sometimes refers to the Ada *Language Reference Manual* [RM 83] as either the *RM* or the *Standard*. The phrase *pragmas and attributes* should be understood to mean the set of all *language-defined* (sometimes referred to as *predefined*) and *implementation-defined* pragmas and attributes.

1.3. Acknowledgments

This report would not have been successful without the support of the Ada Validation Facility at Wright-Patterson Air Force Base, Ohio. The author is grateful to Dale Lange for his help. In addition, the author acknowledges the following staff at the Software Engineering Institute for their time and effort in reviewing the technical content of this study:

John Goodenough
Nelson H. Weiderman
Neal Altman

The following individuals also offered helpful hints in the preparation and improvement of this report:

Pat Donohoe
Timothy Coddington

2. Conclusions and Recommendations

Based on our analysis of the 143 compiler implementations considered in this study, the following issues should be considered in the Ada 9X revision:

- Clarify the support required and provided for language-defined pragmas.
- Extend the functionality of pragma INTERFACE.
- Lessen the dangers of usage-restrictive pragmas.

These recommendations are discussed below.

2.1. Inadequate Support for Language-Defined Pragmas

This study is based primarily on the Appendix F from each compiler Validation Summary Report, but secondary sources, primarily the documentation provided for eight compilers installed at the SEI, were also used when available. The information available from the product manuals of the compilers installed at the SEI added immeasurably to the study since the eight compilers[2] were root compilers for 116 implementations in the sample (82% of the sample) and since the Appendix F for some vendors contained only scanty information compared to the information available in the product manuals. The results of this study showed that:

1. Fewer than 5% of the implementations support all the language-defined pragmas.
2. Fewer than 30% of the implementations documented their support for language-defined pragmas in Appendix F.
3. Fewer than 45% of the implementations gave more than a perfunctory description of implementation-defined pragmas and attributes Appendix F.[3]

The following course of action is recommended:

- The Standard should require that a compiler's implementation-dependent properties be described clearly in Appendix F. To provide guidance to implementors, Appendix F should provide a detailed format to be used in describing these properties. In particular, a format should be specified for describing an implementation's support for all pragmas (both language-defined and implementation-defined) and for implementation-defined attributes.

[2]Telesoft, Verdix, Meridian, System Designers, Tartan Laboratories, Rational, DEC, DDC, and Alsys.

[3]Reference manuals did a much better job in this regard, although even here communications with the vendor were often required to firmly establish the behavior of some implementation-dependent features.

2.2. Extending Pragma INTERFACE

A majority of the surveyed Ada implementations found pragma INTERFACE to be insufficient for user needs. This conclusion is based on the large number of implementation-defined pragmas that replace it. A large percentage of Ada compilers implement pragma INTERFACE as specified by the Standard, but then recommend that customers use a new pragma for the functions originally intended for INTERFACE. The following recommendation is made to redress this problem:

- Extend the functionality of pragma INTERFACE to include the following:
    - the ability to specify an arbitrary external (link-time) name that corresponds to an Ada entity named in the pragma; the external name need not follow the rules for an Ada identifier.
    - for subprograms, the ability to specify the mechanism used to pass parameters; at least by-value and by-reference mechanisms should be specifiable.
    - the ability to specify a correspondence between a declared object and an externally allocated object. (This capability is analogous to the current pragma INTERFACE capability for subprograms.)
    - the ability to make Ada subprograms and declared objects available for use from *outside* the Ada program (the ability to *export* such entities).

While some implementations provide additional interfacing capabilities (e.g., for exceptions as well as objects and subprograms), the set of capabilities listed above will suffice to cover a great number of implementation-dependent pragmas available today.

2.3. Enhancing Performance by Restricting Usage

A class of implementation-defined pragmas in the sample are aimed at reducing execution time or object code size by restricting the use of specific Ada features, thus reducing the overhead associated with supporting these features in full generality at runtime. The restrictions and associated benefits are generally summarized unhelpfully in Appendix F. However, the restrictions and their intents can be summarized as follows:

- <u>To reduce hardware interrupt response time by limiting the use of tasking features within an interrupt entry or task body.</u>

  The Standard defines in RM 13.5.1 how interrupts may be associated with task entries for the purpose of handling hardware signals. Many implementations severely limit the form that a task body may take when interrupts are to be processed. Such limitations allow interrupt entries to be implemented as ordinary procedure calls, without any runtime scheduler intervention, thus reducing the runtime overhead considerably. However, even when targeting the same processor, implementations vary widely in the restrictions imposed on task bodies.

- <u>To reduce object code size by forbidding use of the VALUE and IMAGE attributes for enumeration types.</u>

  The VALUE and IMAGE attributes provide a mapping between the source code representation of an enumeration value and its runtime representation. If these attributes are used, a table must be generated and loaded at runtime that provides the appropriate mapping. If the user does not intend to use these attributes (or perform

TEXT_IO) on enumeration types, then space can be saved by not loading these tables in memory. The point at which the optimization has full effect varies from implementation to implementation and depends heavily on the user following the implementation's rules.

- To reduce record initialization time when whole record comparisons are not performed.

  If a record's representation contains unused bits between components and the record is used in a comparison or equality operation, then an implementation can do the comparison more efficiently if the padding bits are always set to the same value, usually zero. Consequently, some implementations zero out the storage area occupied by such objects to ensure the padding bits have the correct values. Since the padding bits must be zeroed each time the object's declaration is elaborated and each time a value of the type is created, the clearing of memory can have a significant cumulative impact on performance. If a user agrees not to perform composite comparisons, then program execution time can be reduced by no longer initializing these memory locations.

- To reduce subprogram call overhead when there are no recursive or reentrant calls.

  Since Ada subprograms can be called recursively and reentrantly, the local data for each subprogram is allocated on a stack. The allocated storage is freed when a subprogram completes its execution. The overhead of allocating and deallocating stack storage can be avoided if an implementation knows that a subprogram is never called recursively or reentrantly. In general, an implementation cannot determine whether a separately compiled subprogram is ever called recursively or reentrantly, so it must make a conservative assumption and allocate storage on the stack. Even if the implementation knows that recursion is impossible, users may still wish to use stack allocation as a means of reducing overall storage requirements at runtime. So some implementations allow programmers to specify when local storage is to be allocated statically. In these cases, the programmer must not call such subprograms recursively or reentrantly.

There are other varieties of restrictive pragmas as well, but these cover the most important capabilities. Note that these pragmas are not in the same category as the language-defined pragma OPTIMIZE, since no such usage-restrictive behavior is prescribed for this pragma (see RM Annex B).

Implementation-defined pragmas that enhance runtime performance by restricting uses of language features require a careful analysis of the possible long term (life-cycle) effects of this approach. There are a number of possible disadvantages to such pragmas:

1. If a pragma's usage restrictions are violated, the behavior of a program may well be unpredictable, i.e., erroneous. If an implementation detects such a violation, it is not allowed to reject the program (since an implementation-defined pragma cannot have an effect on program legality), but programmers should nonetheless be warned that the expected program behavior will not be achieved.
2. The use of such pragmas affects program portability, since implementations not recognizing the pragmas may not provide the expected optimizations, leading to unacceptable performance behavior.
3. Modifications to a program later in its life cycle may unintentionally violate the restrictions imposed by such a pragma, leading to unpredictable program behavior that may not be immediately detected during acceptance testing.

Given the advantages and dangers of usage-restrictive pragmas, this report recommends that:

- Implementations be allowed, but not encouraged, to continue offering performance enhancement pragmas that are usage-restrictive.
- Implementations uniformly provide an operational mode in which the presence of usage-restrictive pragmas is noted in messages issued at compile-time. Moreover, when operating in this mode, programs may be rejected if any violations of the usage-restrictions are detected by the implementation.[4]
- If an implementation-dependent pragma has no effect, i.e., if it is ignored, then a warning message should be issued to the user.

[4]Note that an implementation may not always be able to detect such violations. For example, if a pragma forbids recursive calls, such calls can, in general, be detected only by analyzing the complete call graph of a program, and even then, some false warnings might be issued.

3. Analysis

This survey of implementation-dependent pragmas and attributes analyzed 143 validated Ada compilation systems. The primary source of data was the Appendix F required from each Ada compiler for government validation. The Appendix F information was obtained from Validation Summary Reports (VSRs) received from the Ada Validation Facility (AVF) at Wright-Patterson AFB, Ohio. This source was supplemented by product manuals, written correspondence, and verbal discussions with various compiler vendors and users. Ninety-five percent of the VSRs obtained for this study were for implementations validated from February through September of 1988 under ACVC 1.9. The remaining 5% of the VSRs were for implementations validated under ACVC 1.10. These systems were either upgraded versions of Ada compilers already installed at the SEI or those received just in time for inclusion in the study. The VSRs representing implementations installed at the SEI, while totaling only 8, were representative of a far greater number of Ada compilers because these implementations served as root compilers for almost 82% of the sample. The SEI compilers provided significant insight for this study because we had access to the more detailed documentation typically provided to purchasers of compilers.

At the end of the VSR sampling period (September 1988), the total number of validated compilers stood at 210. By January 1989, that total had grown to 229, and by the end of May it had reached 258. This increase represents at least a 30% annual growth rate [AIC 89]. This great expansion in the number of Ada implementations means that the study was aimed at a moving target, so its conclusions may be dated. Nonetheless, the sample reflects what can fairly be called a representative fraction of compilers available from October 1988 through March 1989. Due to inadequacies in the content of Appendix F documentation,[5] one compiler was excluded from the survey.

Since compilers must recognize pragmas during front-end syntax processing, similarities among implementation-defined pragmas indicate when compilers for given target processors share the same front end. Based on such evidence, apparently 16 implementations served as root compilers for a much larger number of code generators. The existence of the 16 root compilers

[5]This is an unfortunate consequence of a policy in which the Ada *Reference Manual* (RM) leaves largely undefined the level of required implementation support for language-defined pragmas and attributes. Implementations that (1) abide by the text of the RM exactly, (2) offer no implementation-defined pragmas or attributes of their own for inclusion within Appendix F, and (3) were not installed at the SEI, also tend to provide no useful information for this report.

simplified the overall survey because it induced a common grouping on otherwise disparate implementations.

Approximately one-third of the sample compilers were cross-compilers, which closely matches the proportion of cross-compilers within the total population (38% of all compilers are cross-compilers). Embedded target systems were 30% of the total sample, a significant subset.

Figure 3-1 illustrates the extent of the survey's coverage of Ada implementations, with a breakdown of root compilers by vendor as a percentage of the sample. Of the 16 root compilers, 2 comprise fully 60% of the sample. The next 9 vendors, with from 2% to 8% each, total another 35% of the sample. The last 5% (*others*) refers to 5 root compilers (8 implementations).

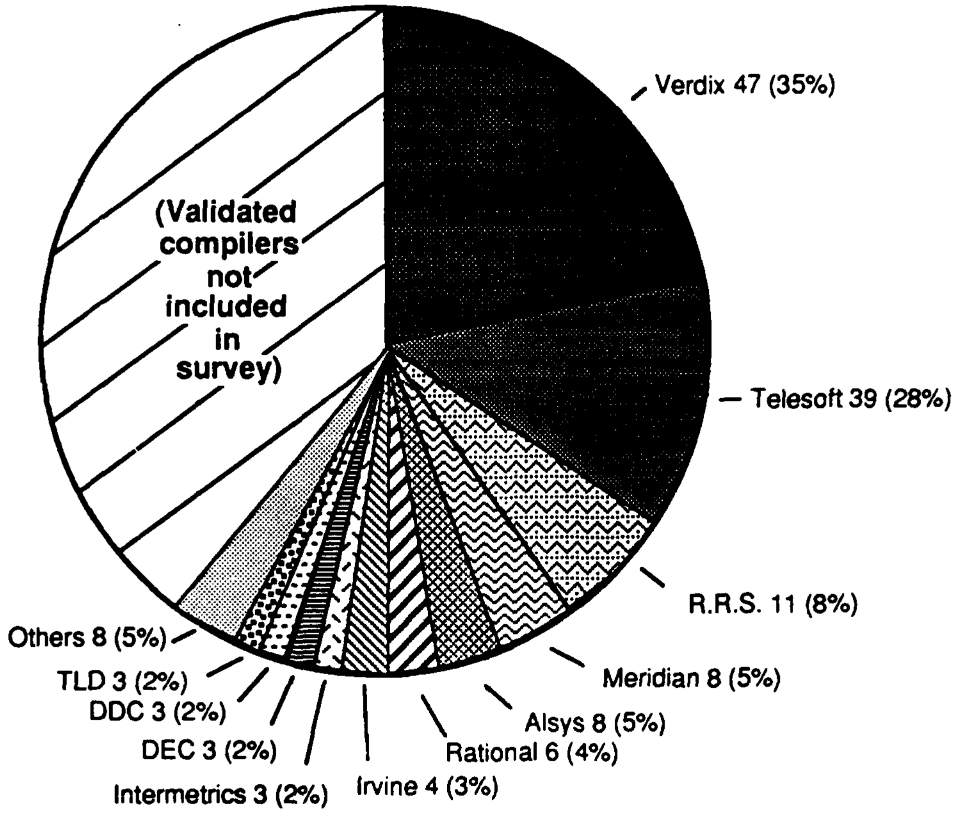
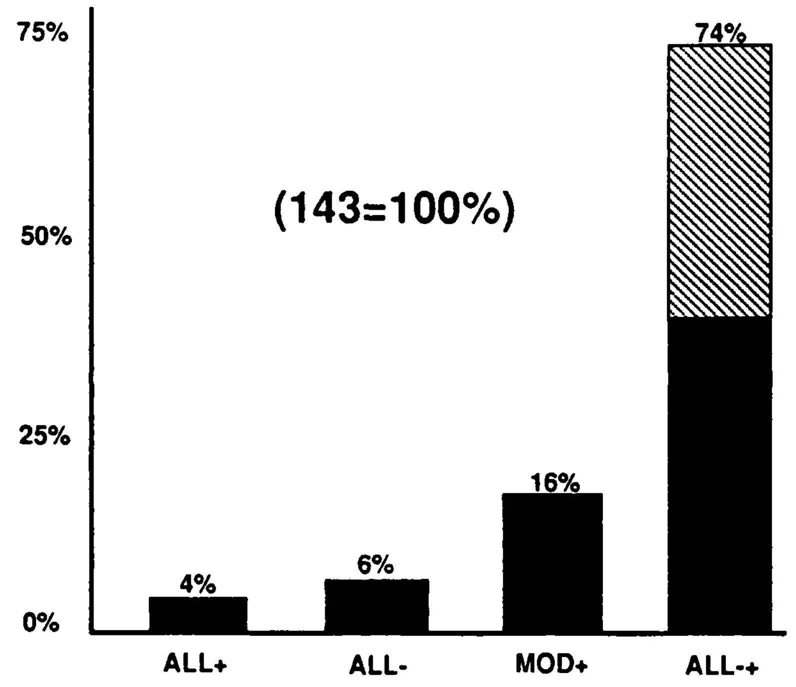


Figure 3-1: Distribution of Root Compilers in the Sample

3.1. Support for Language-Defined Pragmas

Of the 143 implementations surveyed in this report, Figure 3-2 illustrates the extent to which the surveyed implementations indicated support for the language-defined pragmas and for additional implementation-defined pragmas. Four possible conditions are depicted, with raw totals in parenthesis. The largest grouping (**ALL-+**) contains a disproportionate number of derived compilers; these are shown to illustrate their contribution within the sample.

ALL+	Implementations that support <u>all</u> language-defined pragmas plus implementation-defined pragmas[6] (6)
ALL-	Implementations that support a <u>proper subset</u> of the language-defined pragmas (8)



<u>Legend</u>

▨ derived implementations

ALL+	All language-defined pragmas are supported and implementation-defined pragmas are provided.
ALL-	A proper subset of the language-defined pragmas are supported and no implementation-defined pragmas are provided.
ALL-+	A proper subset of the language-defined pragmas are supported and implementation-defined pragmas are provided.
MOD+	A modified subset of the language-defined pragmas are supported and implementation-defined pragmas are provided.

Figure 3-2: Overall Support for Language-Defined Pragmas

[6]The language-defined pragmas (14 in number) are: CONTROLLED, ELABORATE, INLINE, INTERFACE, LIST, MEMORY_SIZE, OPTIMIZE, PACK, PAGE, PRIORITY, SHARED, STORAGE_UNIT, SUPPRESS, SYSTEM_NAME.

MOD+	Implementations that support a modified subset of the language-defined pragmas plus implementation-defined pragmas (23)
ALL-+	Implementations that support a proper subset of the language-defined pragmas plus implementation-defined pragmas (106)

Figure 3-3 shows the extent of support for each language-defined pragma within all surveyed systems. The implementations are grouped by common front ends; this categorization was based on a study of similarities among the Validation Summary Reports and is not necessarily confirmed by the individual vendors. The top to bottom ordering of pragmas is by observed support level, from most to least The left to right ordering of root compilers is also by observed level of support, from most to least.

The most frequently unsupported pragmas are SYSTEM_NAME, STORAGE_UNIT, and MEMORY_SIZE. The OPTIMIZE pragma shows only a smattering of support, with a great many vendors opting to provide a compiler switch[7] as a more useful alternative for expressing choice. When an optimization pragma *is* provided, it usually provides a finer granularity of control than just TIME or SPACE.

The apparent lack of support for pragma CONTROLLED is probably the result of a misunderstanding on the part of vendors. When an implementation does not provide automatic storage reclamation for collections, pragma CONTROLLED is always implicitly in effect. Since its explicit use has no additional effect, some vendors apparently claim they do not support the pragma.

The only standard pragma that is always supported is pragma ELABORATE (since support for this pragma is tested in the validation process) followed by, in close order, support for pragmas INTERFACE, INLINE, PAGE, LIST, and PACK. There are also a few cases where pragma PRIORITY is trivially supported with, for example, a priority range of 0..0. Pragma INTERFACE, while widely supported, was also apparently found to be insufficient for most user needs due to the number and diversity of implementation-defined extensions.

The sample shows wide support for pragmas PAGE and LIST, but a few vendors thought listing control was better provided by a compiler switch.

Two common modifications of pragmas SUPPRESS and INTERFACE were observed: for SUPPRESS, disallowing the argument giving the name of a program entity, and for pragma INTERFACE, allowing additional arguments specifying the link-time names of subprograms.

In general, most implementations found the existing definition of pragma INTERFACE to be inadequate. Alteration of the pragma to increase its effectiveness usually took the form of an added argument association to indicate an external name. Even this relatively benign alteration is insufficient to meet all user needs (see Section 3.2.1 in this report).

[7]Host operating system command line option.

Ada Compiler Implementation by Root / Language-Defined Pragma	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	RA	TT	VE	ME	IC	TE	IN	DD	TA	TL	AL	DE	C3	SD	HA	RR
I	6	2	47	8	4	39	3	3	1	3	8	3	1	1	2	11
C	4	2	16	0	1	14	0	2	0	3	1	2	0	1	0	0
D	0	0	30	0	0	10	0	0	0	0	4	0	0	0	0	0
Elaborate	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩
Interface	▩	□	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	□
Page	▩	▶	▩	▩	▩	□	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩
Inline	▩	▩	▩	▩	▩	▩	▩	▩	▩	□	▩	▩	▩	□	▩	□
Pack	▩	▩	▩	▩	▩	▩	▩	▩	□	▩	□	▩	▩	▩	▩	□
List	▩	▶	▩	▩	U	□	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩
Priority	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	▩	□	▩	U	□
Shared	▩	▩	□	▩	▩	▩	□	▩	▩	▩	□	□	□	□	U	▩
Suppress	▩	◩	▩	▩	▩	▩	◪	◪	□	▩	▩	□	□	□	□	◪
Controlled	▩	▩	□	▩	▩	▩	▩	□	▩	□	□	□	□	□	□	□
Optimize	▩	□	□	▩	▩	▩	□	□	□	□	□	□	□	□	□	▩
System_Name	▩	▩	▩	□	□	◩	□	□	□	□	□	□	□	□	□	□
Storage_Unit	▩	▩	▩	□	□	◩	□	□	□	□	□	□	□	□	□	□
Memory_Size	▩	▩	▩	□	□	◩	□	□	□	□	□	□	□	□	□	□

Legend

AL -	ALSYS, Inc.	RA -	Rational	▩	Fully supported
C3 -	Concurrent Computer Corporation	RR -	R.R. Software, Inc.	◩	Fully supported on some implementations, but unsupported on others
DD -	DDC International	SD -	System Designers, Inc.	◪	Supported with restrictions or alterations on all implementations
DE -	Digital Equipment Corporation	TA -	Tandem Computers	▶	Fully supported on some implementations, restricted support on others
HA -	Harris Corporation	TE -	TeleSoft, Inc	□	Not supported
IC -	Irvine Compiler Corporation	TL -	TLD Systems, Inc.	U	Not determined
IN -	Intermetrics	TT -	Tartan Laboratories		
ME -	Meridian	VE -	Verdix Corporation		

I - total number of implementations (validations + derived implementations)
C - number of cross-compilers (I - self-hosted implementations)
D - number of derived implementations (validated by derivation from previously certified compiler)

Figure 3-3: Support for Language-Defined Pragmas, by Implementation

3.2. Implementation-Defined Pragmas

There are approximately 110 implementation-defined pragmas offered by the various compilers in this survey. An analysis classifying each by functionality reduces this number to 20 or so groups.[8] Some vendors add functionality to existing pragmas, while others provide new pragmas that are intended to replace the language-defined ones. When a replacement is provided, some implementations do not continue to support the replaced pragma. The remaining implementation-defined pragmas address unique concerns specific to a target processor or programming environment.

The following functional classes of pragmas were found. The classes are listed in approximate order of breadth of support.

- Import/Export Pragmas
- Non-restrictive Performance Enhancement Pragmas
- Usage-restrictive Pragmas
- Checks Suppression
- Conditional Compilation
- Information Control
- Target Resource Control
- Target Specific Operations
- Miscellaneous

Examples of pragmas belonging to each class are given in subsections later.

Figure 3-4 presents the breadth of support for these pragmas on several dimensions: by breadth of vendor support (percent coverage of surveyed root compilers), by potential market penetration (percent coverage of all surveyed implementations), and by cross-compiler influence (percentage of cross-compilers which contribute to each grouping.) The figure is sorted left to right by degree of root compiler support, greatest to least.

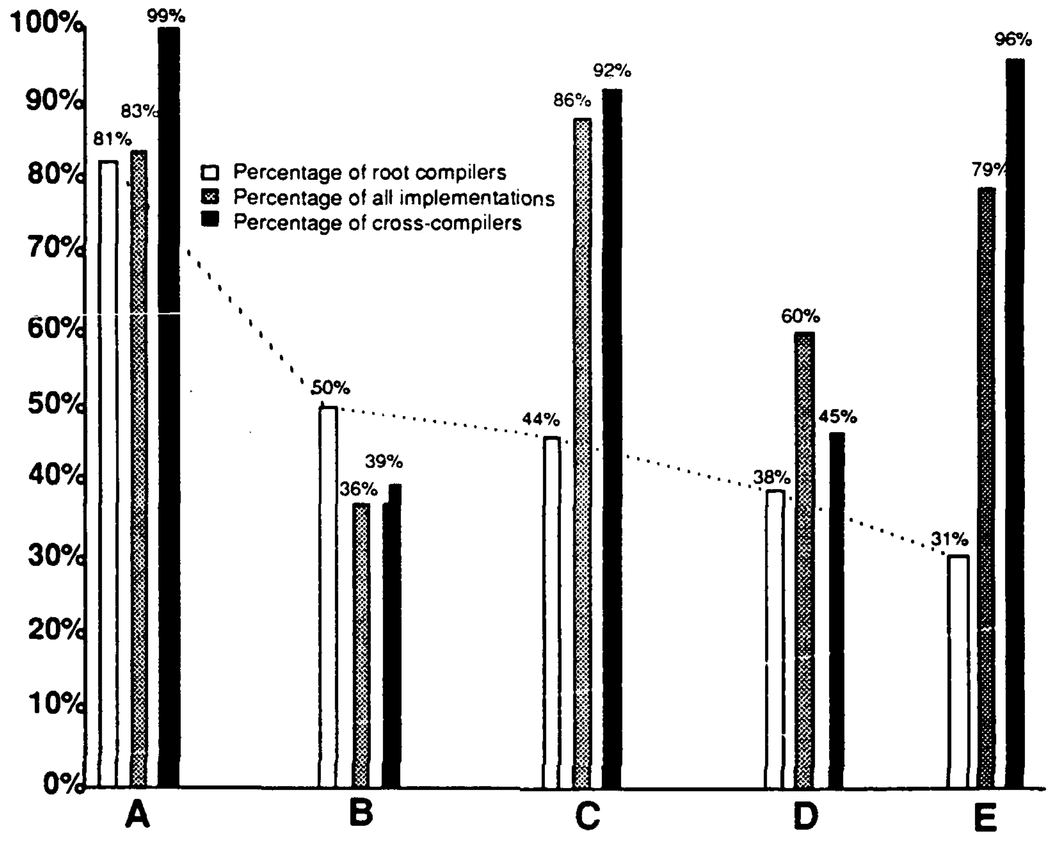
3.2.1. Import/Export Pragmas

Ada applications sometimes need to be linked with non-Ada programs. While it is traditional to think of Ada calling a foreign unit, this is not always the case. Examples of calls from external sources to Ada programs are not frequent but can arise when an existing non-Ada application is modified by code written in Ada, or when subsystems need to call Ada programs to provide special services. (For example, the proposed X-Window interface specifies certain subprograms that are to be provided by the Ada application and that will be called by the X-Window routines.) To provide external visibility to such callers, implementations provide an EXPORT pragma such as the following:

```
        pragma EXPORT ( Ada_name, string_literal );
```

The flip side of the coin allows Ada programs to access externally provided services using a

[8]See also the summary provided by Nelson Weiderman in Section 5.1.2 of [ACS 89].



Legend

Total by category and percentage ():

A	- Language-Defined Substitute Pragmas[1]	44	(39%)
B	- Host-Specific Pragmas[2]	23	(20%)
C	- Performance Enhancement Pragmas[3]	10	(9%)
D	- Target-Specific Pragmas[4]	21	(18%)
E	- Usage-Restrictive Pragmas[5]	16	(14%)
		114	

Example: 44% of the root compilers (7 of 16) defined 10 pragmas for program performance enhancement. In the same category we found that these roots represented 86% of all implementations and 92% of all cross-compilers in the sample.

Notes:

1 - ***Import/Export (31), Checks Suppression (8), Volatile (3), Optimize (1)***

2 - ***Information control; conditional compilation***

3 - ***Sharing of generic bodies, machine-code insertion optimizations, compact object representations***

4 - ***Target resource allocation, target runtime control and target environment configuration, test, and debug***

5 - ***Performance enhancements subject to voluntary use of a subset of the language***

Figure 3-4: Compiler Support for Implementation-Defined Pragmas

name that may not satisfy Ada's syntax for identifiers. For example, a foreign subprogram name can be associated with a subprogram named previously in a pragma INTERFACE.

```
    pragma IMPORT ( Ada_name, string_literal );
```

Note that in both examples above the argument associations are identical. In either case an Ada subprogram specification acts as the visible link to the outside world. When a foreign unit is imported, the body of the named Ada unit does not exist. When the Ada unit is exported, then an Ada subprogram body must be supplied to perform the intended operation. In both cases an external link name is needed. For further detailed explanations from each implementation refer to INTERFACE_NAME, INTERFACE_SPELLING, EXTERNAL_NAME, LINKAGE_NAME, NICK-NAME, FOREIGN_BODY, EXPORT, and IMPORT. (The Index specifies the page on which each pragma is discussed.)

A frequent feature in these import/export schemes is the ability to specify objects in addition to subprograms. For examples of this capability, refer to INTERFACE_OBJECT, PSECT_OBJECT, IMPORT_OBJECT, EXPORT_OBJECT, IMPORT, and EXPORT. Also, some compilers offered interface facilities for an optional description of the physical representation of subprogram parameters (such as object descriptors[9] and the parameter passing mechanism).[10] Examples of this feature can be seen in more detail under P'NULL_PARAMETER, P'TYPE_CLASS, CONDITION_CODE, EXTENSIBLE_ARGUMENT_LIST, VARIABLE_ARGUMENT_LIST, IMPORT_FUNCTION, IMPORT_PROCEDURE, IMPORT_OBJECT, IMPORT, EXPORT_FUNCTION, EXPORT_PROCEDURE, EXPORT_OBJECT, INTERFACE_COMMON_OBJECT, INTERFACE_OBJECT, PSECT_OBJECT and EXPORT.

```
    pragma EXPORT_FUNCTION( INTERNAL => function_name,
                            EXTERNAL => string_literal,
                            PARAMETER_TYPES => type_mark,
                            RESULT_TYPE => type_mark,
                            MECHANISM => value );
```

3.2.2. Non-Restrictive Performance Enhancement Pragmas

Non-restrictive performance enhancement pragmas do not, in general, place significant restrictions on the user with regard to use of the language, and can include the following enhancements:

- Sharing of generic bodies. Refer to SHARE_BODY and SHARE_CODE for greater detail.

  ```
  pragma SHARE_BODY( generic_name, boolean_literal );
  ```

- Data compression techniques applied to specific objects. Refer to BIT_PACK, COMPRESS, and SQUEEZE for greater detail.

[9]Linker composite representations which describe the physical characteristics and location of a data element.

[10]Physical method of transfer, e.g., VALUE, REFERENCE, DESCRIPTOR (as in DEC Ada).

```
type type_name is
   record
      component_list
   end record;
pragma SQUEEZE( type_name );
```

- Manipulation of machine-code insertions. Refer to IMPLICIT_CODE and OPTIMIZE_CODE for greater detail.

```
with MACHINE_CODE;
procedure CLEAR_DEVICE;
pragma INLINE( CLEAR_DEVICE );
pragma IMPLICIT_CODE( OFF );
procedure CLEAR_DEVICE is
  use MACHINE_CODE;
begin
  coded_instruction_list
end;
```

- Selection of global optimization passes or specific optimization techniques. Refer to OPT_LEVEL, OPTIMIZE_CODE, and INLINE_ONLY for greater detail.[11]

```
pragma OPT_LEVEL( GLOBAL );
```

- Informing the compiler of shared variables to exempt them from various optimizations.[12] See VOLATILE for greater detail.

```
type R is
  record
     component_list
  end R;
Composite : R;
pragma VOLATILE( Composite );
```

The distinction between pragmas SHARED and VOLATILE is that SHARED can only be used with scalar or access types [RM 9.11(10)]. VOLATILE objects may be composite (record or array). See also the forthcoming companion Ada 9X study on the treatment of shared variables in Ada.

3.2.3. Usage-Restrictive Pragmas

Pragmas of this nature require the user to avoid certain constructs of the Ada language. In return, these pragmas either provide performance enhancements or certain machine-dependent functions. The usage restrictions may be explicit, i.e., stated by an implementation as a requirement for use, or they may be implicit in the use of the pragma. The following examples typify these pragmas:

- Associate external device interrupts with entries of an application task. Refer to AST_ENTRY, INTERRUPT, LEVEL, PASSIVE, and INTERRUPT_HANDLER for greater detail.

[11]As with the language-defined pragma OPTIMIZE, the majority of implementations prefer a host operating system command-line switch option for global optimization rather than a pragma in the source text. The use of switches is "user friendly" in that the choice can be made without taking the chance of introducing an error in the program text. On the other hand, the use of a switch provides no visible evidence in the source text of the optimization request.

[12]This can include assignment killing, equivalence propagation, strength reduction, code movement, and the hoisting of invariant expressions, among others.

```
task task_name is
  pragma INTERRUPT_HANDLER;
  entry InterruptEntry_name;
end;
```

- Disable initialization of storage units during the elaboration of a declaration. This feature reduces elaboration time but precludes the user from performing certain subsequent operations including composite object comparisons. Refer to NO_ZERO for greater detail.

```
type type_name is
record
  component_list
end record;
...
pragma NO_ZERO( type_name );
```

- Eliminate runtime table generation for enumeration types. This feature reduces the object file size at the cost of preventing the user from using the IMAGE, VALUE, and WIDTH attributes for these types. Refer to IMAGES, NO_IMAGE, RESTRICTED_ENUMERATIONS, and ENUMTAB for greater detail.

```
type type_name is ( identifier_list );
...
pragma IMAGES( type_name, Deferred );
```

- Prohibit recursive and reentrant calls to reduce overhead for allocating data. This feature reduces execution time at the possible cost of STORAGE_ERROR. The user is warned of excessive nesting and of the use of recursion. Refer to CLEANUP and NON_REENTRANT for greater detail. Inlining optimizations may also be enabled.

```
pragma CLEANUP( 0 ); -- no stack space recovered
```

- Improve optimization of subprogram calls by eliminating external dependencies. If subprograms make no external references, redundant calls can be eliminated. Refer to NO_SIDE_EFFECTS for greater detail.

- Reduce runtime space created by some implementations when subprograms are derived. This pragma reduces object code space by forbidding derived types. See RESTRICTED_SUBPROGRAMS and RESTRICTED_RECORDS for greater detail.

- Eliminate elaboration code. Code for elaborating packages can be eliminated if certain user restrictions are obeyed. See NOT_ELABORATED and SHARED_PACKAGE for greater detail.

3.2.4. Checks Suppression

Suppress all or certain groupings of runtime checks (8 instances). Typically, a single pragma is provided that has the effect of suppressing all predefined checks. Refer to ARITHCHECK, RANGECHECK, STACK_CHECK, ALL_CHECKS, SUPPRESS_ALL and SUPPRESS_STACK for greater detail.

```
pragma SUPPRESS_ALL;  -- Declared at front of compilation unit.
                      -- Suppresses all checks in scope of unit.
```

3.2.5. Conditional Compilation

This group of functions requires a pre-compiler pass over the source text to accomplish the requested effect.

- Structured commands, akin to the Ada If Statement, to govern text compilation. Refer to BEGIN_COMPILE, NOW_COMPILE, IF, ELSE, ELSIF, END, END_COMPILE, and STOP_COMPILE for greater detail.

```
-- begin conditional-compilation block

pragma IF (compile_time_boolean_expression);
   ... -- program source text; compile when expression = TRUE
   ...
   ...
pragma ELSE;
   ... -- alternative text; compile when expression = FALSE
   ...
   ...
pragma END;  -- end conditional compilation block
```

- Directives for incorporating a source file for compilation at the place of the pragma. Refer to INCLUDE (three instances) in Index for greater detail.

```
pragma INCLUDE( file_path_name_string_literal );
```

3.2.6. Information Control

A number of pragmas in the sample allow user control over the amount of information to be produced at compile-time or for unhandled exceptions at runtime.

- Commands for generating compile-time output. Refer to PUT and PUT_LINE for greater detail.

```
pragma PUT_LINE( record_type );
```

- Pretty-printing format. Refer to INDENT for greater detail.

```
pragma INDENT( ON );  -- turn pretty-printer control on
```

- Debugger output. Refer to DEBUG (two instances) for greater detail.

```
pragma DEBUG( ON );
pragma DEBUG( NAME => value );
```

- Pagination. Refer to PAGE_LENGTH for greater detail.

```
pragma PAGE_LENGTH( integer_literal );
```

- Insert text in the object file. Refer to COMMENT for greater detail.

```
pragma COMMENT( string_literal );
```

- Exception traceback. Refer to VERBOSE for greater detail.

```
pragma VERBOSE( OFF );
```

3.2.7. Target Resource Control

Several pragmas are provided for controlling the location of data in physical memory. Examples are as follows:

- Set a library task stack memory segment size. Refer to LT_SEGMENT_SIZE for greater detail.

```
pragma LT_SEGMENT_SIZE( task_type_mark, integer_expression );
```

- Set a task's process stack size (as opposed to the activation storage size). Refer to STACK_SIZE for greater detail.

```
pragma STACK_SIZE( task_type_mark, integer_expression );
```

- Set a task's guard page storage size. Refer to TASK_STORAGE for greater detail.

```
pragma TASK_STORAGE( TASK_TYPE => task_type_mark,
                     TOP_GUARD => static_simple_name );
```

- Set base page physical memory loading of selected program units or objects. Refer to PRIMARY and PAGE_LIMIT in Index for greater detail.

```
pragma PAGE_LIMIT( integer_expression );
```

3.2.8. Target Specific Operations

There exist a number of pragmas which provide operations in the context of a tightly integrated development environment, specific underlying operating system services (beneath the Ada run-time kernel), or in support of an unusual application domain with appropriate target hardware.

- Enable and disable automated storage reclamation. Refer to ENABLE_DEALLOCATION and DISABLE_DEALLOCATION for greater detail.

```
pragma ENABLE_DEALLOCATION( type_name );
```

- Provide a vendor with proprietary maintenance "hooks" into the target system. Refer to BUILT_IN and RESTRICTED_OPERATORS for greater detail.

```
pragma BUILT_IN;
```

- Designate a program unit as vectorizable or force to scalar, assuming an underlying vector architecture. Refer to SCALAR, NO_RECURRENCE and VECTOR_UNIT for greater detail.

```
pragma SCALAR;
```

- Interface objects to a common monitor (built-in debug) target memory section. Refer to INTERFACE_MCOM_OBJECT for greater detail.

```
pragma INTERFACE_MCOM_OBJECT( simple_name,
                              string_literal,
                              string_literal );
```

- Assign characteristics to a program unit which govern its behavior independent of the Ada runtime effect (e.g., as in a Rational SubSystem). Refer to LOADED_MAIN, MAIN, MUST_BE_CONSTRAINED, OPEN_PRIVATE_PART, PRIVATE_EYES_ONLY and SYSLIB for greater detail.

```
pragma MUST_BE_CONSTRAINED( conditional_expression, type_name );
```

3.2.9. Miscellaneous

Here are some examples of pragmas that do not fit into any of the above categories:

- Selection of underlying hardware physical representation for numeric data types. Refer to LONG_FLOAT for greater detail.

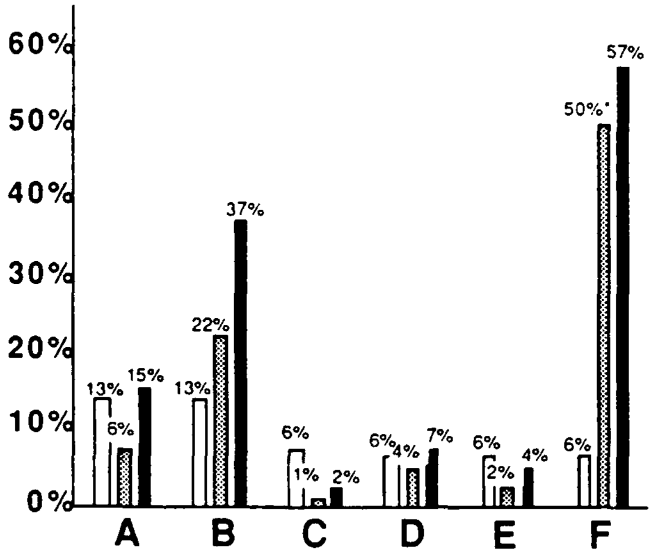
```
pragma LONG_FLOAT( D_FLOAT );
```

- Selection of underlying kernel timing behavior (e.g., as in the choice of task time slice duration). Refer to TIME_SLICE for greater detail.

```
pragma TIME_SLICE( static_expression );
```

3.3. Implementation-Defined Attributes

The total number of implementation-defined attributes is quite small. This survey counted only 18, compared to more than a hundred pragmas. With such a small number of attributes, it is difficult to draw any conclusions, yet certain functional characteristics were observed.



Implementation-Defined Attributes

Legend

A - Memory Allocation	2
B - Object Alignment	2
C - Object Address	3
D - Object Description	3
E - Target Configuration	3
F - Miscellaneous	5
	18 total

□ Percentage of root compilers
▨ Percentage of all implementations
■ Percentage of cross-compilers

* Machine Code Insertion Attribute by single vendor with significant number of derived licenses.

Figure 3-5: A Functional Profile of Implementation-Defined Attributes

Figure 3-5 shows the breadth of support for these attributes on several dimensions: by breadth of vendor support (percent coverage of surveyed root compilers), by potential market penetration (percent coverage of all surveyed implementations), and by cross-compiler influence (percentage of cross-compilers which contribute to each grouping.) The figure is sorted left to right by degree of root compiler support, greatest to least.

The following functional groupings were observed:

- Object Properties

- Target Configuration
- Miscellaneous

3.3.1. Properties of an Object

There are several attributes that allow a program to inquire about the properties of an object or type.

- Yield the number of STORAGE_UNITS allocated to a defaulted variant record. Refer to P'RECORD_SIZE for greater detail.
- Yield the actual number of bits to be allocated to an object of the designated type, including padding. Refer to P'MACHINE_SIZE for greater detail.
- Yield the offset of the first bit in the first storage unit allocated for an object. Refer to P'BIT for greater detail.
- Yield the offset equal to the address difference between the first storage unit allocated to an object and the page base register in which the object is located. Refer to P'OFFSET for greater detail.
- Yield different forms of address for an object. Refer to P'EXTENDED_ADDR, P'WORD_ADDRESS, and P'STRING_ADDRESS for greater detail.
- Yield the current discriminant index or descriptor of an optimized record or array. Refer to P'VARIANT_INDEX, P'RECORD_DESCRIPTOR and P'ARRAY_DESCRIPTOR for greater detail.

3.3.2. Target Configuration

There are several attributes that allow a program to query the target configuration:

- Yield a value specifying the target architecture and runtime system. Refer to P'VERSION, P'SYSTEM, and P'TARGET for greater detail.

  ```
  discrete_type_name'VERSION -- target environment version identifier
  ```

3.3.3. Miscellaneous

Here are some examples of attributes that do not fall into any of the previous classes:

- Yield the machine code insertion operand. Refer to P'REF for greater detail.

  ```
  constant_variable_subprogram_label_name'REF -- code statement operand
  ```

- Yield the shared-package built-in semaphore status while requesting access or release of resource. Refer to P'LOCK and P'UNLOCK for greater detail.

  ```
  package_name'LOCK( boolean_expression [, integer_expression]) -- P(S)
  package_name'UNLOCK -- V(S)
  ```

- Yield an operating system no-op argument. Refer to P'NULL_PARAMETER for greater detail.

  ```
  type_name'NULL_PARAMETER      -- argument for OS calls
  ```

- Yield an import/export entity type class. Refer to P'TYPE_CLASS for greater detail.

  ```
  type_name'TYPE_CLASS -- class of "full" type of prefix
  ```

- Yield the value of associated interrupt trap. Refer to P'AST_ENTRY for greater detail.

  ```
  task_entry_name'AST_ENTRY -- AST_HANDLER value for OS system services
  ```

4. Sample Findings

This chapter contains detailed information obtained from Ada Validation Summary Reports, Appendix B (Ada implementation Appendix F), and user manuals (in some instances). Each section specifies the support provided by a given vendor's root compiler and its descendent compilers. In some cases, the parentage of a descendent compiler was inferred from the similarity of the implementation-dependent pragmas that were provided, even when the descendent compiler was marketed without any reference to its apparent root compiler.

Information is given for the following vendors:

- Alsys, Inc.
- Concurrent Computer Corporation
- DDC-I, Inc.
- Digital Equipment Corporation
- Harris Corporation
- Intermetrics, Inc.
- Irvine Compiler Corp.
- Meridian Software Systems, Inc.
- Rational, Inc.
- R. R. Software, Inc.
- System Designers, Inc.
- Tandem Computers
- Tartan Laboratories
- Telesoft, Inc.
- TLD Systems, Inc.
- Verdix Corp.

4.1. Alsys, Inc.

Observed in the following compilation systems, except when noted:

Alsys Ada, Version 3.5
DEC MicroVAX II, VMS Version 4.7 host,
Motorola 68020 MVME-133E with MC68881 Floating-Point Co-processor target,
Appendix F

Hewlett Packard HP 9000 Series 300 Ada Compiler, Version 3.25
HP 9000 Series 300 Model 360, HP-UX Revision 6.2 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Hewlett Packard HP 9000 Series 300 Ada Compiler, Version 3.25
HP 9000 Series 300 Model 350, HP-UX Revision 6.01 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Hewlett Packard HP 9000 Series 300 Ada Compiler, Version 3.25
HP 9000 Series 300 Model 310, HP-UX Revision 6.01 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Hewlett Packard derived host/target configurations:

Hewlett Packard HP 9000 Series 300 Ada Compiler, Version 3.25
HP 9000 Series 300 Model 318, HP-UX Revision 6.01 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Hewlett Packard HP 9000 Series 300 Ada Compiler, Version 3.25
HP 9000 Series 300 Model 319, HP-UX Revision 6.01 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Hewlett Packard HP 9000 Series 300 Ada Compiler, Version 3.25
HP 9000 Series 300 Model 320, HP-UX Revision 6.01 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Hewlett Packard HP 9000 Series 300 Ada Compiler, Version 3.25
HP 9000 Series 300 Model 330, HP-UX Revision 6.01 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **CONTROLLED**
- **MEMORY_SIZE**
- **OPTIMIZE**
- **PACK**
- **SHARED**
- **STORAGE_UNIT**

- SYSTEM_NAME

Implementation-Defined Pragmas

INTERFACE_NAME (Ada_subprogram_name, string_literal)

Description: Directive associates an external subprogram identifier, possibly using a non-Ada naming convention, with a valid Ada subprogram. This pragma is used to import subprograms to be called by Ada program units. The pragma is used in conjunction with, and following a pragma INTERFACE. INTERFACE_NAME must appear in the same declarative region as INTERFACE.

INDENT ([OFF | ON])

Description: Directive specifies pretty-printer control of source code indentation. The pragma must be used only with the Alsys Reformatter.

BEGIN_COMPILE
END_COMPILE
NOW_COMPILE
STOP_COMPILE

Description: Presumably these pragmas are used for conditional compilation. After stating the existence of the above pragmas (with the proviso that a full description follows in the succeeding sections of Appendix F), no further explanation is provided.

These pragmas are only valid in the HP 9000 Series host/target implementation.

Implementation-Defined Attributes

P'RECORD_SIZE

Description: If P is a record type with default discriminants, then RECORD_SIZE returns a value equal to the amount of storage allocated by the compiler for a record object of this type. The attribute reflects additional fields of the record due to compiler internal needs.

P'VARIANT_INDEX

Description: If P is a variant record type then VARIANT_INDEX returns a value equal to the current choice of the variant part of a record object of this type. The attribute reflects code changes for more efficient checking of discriminant values.

P'ARRAY_DESCRIPTOR

Description: If P is both a component of an array type and the component subtype is a discriminated record type, then ARRAY_DESCRIPTOR "refers" to the subtypes of those components within the record type which depend on the discriminant(s). The vendor leaves so many particulars about this attribute undefined that it is entirely unclear as to just exactly what is being returned.

P'RECORD_DESCRIPTOR

Description: If P is a component of a record type and the component subtype is itself a record type with a discriminant dependency, then RECORD_DESCRIPTOR "refers" to the subtypes of those components within the record type which are dependent. The vendor leaves so many particulars about this attribute undefined that it is entirely unclear as to just exactly what is being returned.

4.2. Concurrent Computer Corporation

Observed in the following compilation systems, except when noted:

C^3 Ada,
3280 MPS, OS/32 Version R08-02.02 host/target

Derived host/targets:

3200 MPS, OS/32 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **CONTROLLED**
- **MEMORY_SIZE**
- **OPTIMIZE**
- **PRIORITY**
- **SHARED**
- **STORAGE_UNIT**
- **SUPPRESS**
- **SYSTEM_NAME**

Implementation-Defined Pragmas

BIT_PACK (composite_subtype_name)

Description: Directive specifies a reduction in storage for objects of an array or record type. Compression of data is achieved to the bit level. The components of the named object must not be of a floating point type. An (unspecified) increase in compilation time and execution time will result when the pragma is used.

PARTIAL_IMAGE (package_name)

Description: Directive specifies the build of a "partial image" using the named package for this purpose. The compiler verifies that the named package conforms to (unspecified) requirements for using this feature. No further information on this pragma is available from the vendor's Appendix F.

STACK_CHECK ([ON | OFF])

Description: Directive suppresses (when OFF) additional code to check for stack overflow. The pragma applies to subprogram calls and task activations within the program unit containing this pragma. No further information on this pragma is available from the vendor's Appendix F.

SUPPRESS_ALL

Description: Directive suppresses all runtime checks except STORAGE_CHECK. The pragma must appear before each compilation unit.

Implementation-Defined Attributes

No implementation-defined attributes are provided by this vendor.

4.3. DDC-I, Inc.

Observed in the following compilation systems,
except when noted:

DDC-I Ada Compiler System Version 4.3 (ACVC 1.10, March 1989)
DEC VAX, VMS Version 4.7 host,
DACS-80X86, bare machine target.
User's Guide,
Chapter C- Pre-defined Types and Pragmas, and Appendix F.

DDC-I Ada Compiler System Version 4.1.1
DEC VAX 8530 VMS Release 4.5 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

DDC-I Ada Compiler System Version 4.1
DEC VAX 8650 VMS 4.7 host,
CAPS/AAMP SBC (Rockwell International) target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **CONTROLLED**
- **MEMORY_SIZE**
- **OPTIMIZE**
- **STORAGE_UNIT**
- **SUPPRESS** *Modified — second argument not allowed.*
- **SYSTEM_NAME**

Implementation-Defined Pragmas

INTERFACE_SPELLING (Ada_subprogram_name, string_literal)

Description: Directive associates an external subprogram identifier, possibly using a non-Ada naming convention, with a valid Ada subprogram. This pragma is for use by Ada program units which intend to import an external unit for calling purposes. The string literal provides a linker name to be associated with the Ada name. The pragma is used in conjunction with, and following a pragma INTERFACE. Identical to Alsys' pragma INTERFACE_NAME.

This pragma not supported in the CAPS/AAMP SBC compiler.

INTERRUPT_HANDLER

Description: Directive designates a task as an interrupt handler, i.e. as an interrupt service routine (ISR). The interrupt handler must be a task object and the pragma must appear first in the specification of the task. Task entries must be parameterless and cannot be an entry families. The task entries must not be called from any user-defined task. The body of the handler task must not contain anything other than simple accept statements, potentially enclosed in a loop, and must reference only global variables. The "statement list" of a "simple accept statement" is allowed to contain calls to "normal, single, and parameterless" entries of other tasks, but no other tasking constructs are allowed. The call will not cause an immediate context switch, but will be identified as a scheduling point for the called task once control is returned from the interrupt server. No examples are offered. It is somewhat unclear whether the so-called "simple accept statement" and "statement list" refers to an accept statement followed by a sequence_of_statements or to the sequence of statements within the body of the accept. The permissible variety of enclosing loops also is not specified. The ability to schedule a third-party task, within the interrupt handler, is of interest however, since other implementations that provide this feature normally disallow entry calls (see the Telesoft INTERRUPT pragma and the Verdix PASSIVE pragma in the Index).

This pragma is only supported in Version 4.3 compilers.

LT_SEGMENT_SIZE (T, N) *replaces* **LT_STACK_SPACE**

Description: Directive specifies the size of a library task stack segment. T can denote either a task type or a task object. N is the requested size in "words." The segment size determines how many tasks can be created within the library task. The library task stack size can be specified with a P'STORAGE_SIZE representation. The pragma forces all tasks created within the library task to have their stacks allocated from the same segment as the library task. The default stack segment size is the library task stack size. The pragma must be used only for library tasks. The pragma must appear immediately after T's declaration. N must be greater than or equal to the library stack size.

This pragma only supported in version 4.3 compilers, Version 4.2 has a similar pragma titled LT_STACK_SPACE.

EXTERNAL_NAME (name, external_name)

Description: Directive associates an external name with: a permanent object (an object defined in a package specification or body), a constant non-composite object, or a subprogram (whose body is not given by a subunit). This pragma allows Ada entities to be referenced by a non-Ada program. Other naming restrictions apply to the external name.

IMPORT (internal_name [, external_name])

Description: Directive associates an external subprogram identifier with an Ada subprogram. Internal and external names may be either identifiers or string literals. The external name need not have the form of an Ada identifier as long as it is recognizable by the Ada linker.

This pragma is only supported by the CAPS/AAMP SBC compiler.

EXPORT (internal_name [, string_literal])

Description: Directive provides an external name for an Ada subprogram or object. The internal name must be a valid Ada identifier and the external name must be recognizable by the Ada linker. Ada objects must be static. Further restrictions apply to the location of the pragma.

This pragma is only supported by the CAPS/AAMP SBC compiler.

STACK_SIZE (task_type_name, integer_expression)

Description: Directive specifies the number of storage units to be allocated for the process stack of tasks of the associated task type.

This pragma is only supported by the CAPS/AAMP SBC compiler.

Implementation-Defined Attributes

No implementation-defined attributes are provided.

4.4. Digital Equipment Corporation (DEC)

Observed in the following compilation systems,
except when noted:

DEC VAX Ada Version 1.5 (ACVC 1.9)
DEC MicroVAX II, VMS 4.7 host/target.
VAX Ada RM Chapter 13 and Appendix F

System Designers Software Inc. XD Ada Version 1.0 (ACVC 1.10, March 1989)
DEC MicroVAX II, VAX Model 3XXX, 6XXX, 8XXX, VMS Version 5.X host
Motorola 68020 MVME 135, bare machine target
Appendix F

System Designers Software Inc. XD Ada Version 1.0 (ACVC 1.10, March 1989)
DEC MicroVAX II, VAX Model 3XXX, 6XXX, 8XXX, VMS Version 5.X host
MIL-STD 1750A, bare machine target
Appendix F

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **CONTROLLED**
- **MEMORY_SIZE**
- **OPTIMIZE** *(A compiler switch option allows for time or space optimization.)*
- **SHARED**
- **STORAGE_UNIT**
- **SUPPRESS**
- **SYSTEM_NAME**

Implementation-Defined Pragmas

AST_ENTRY (entry_simple_name)

Description: Directive specifies that a task entry may be used to handle a VAX/VMS asynchronous system trap (AST) resulting from a VAX/VMS system service call. The pragma must appear immediately following the entry declaration within the task type or task object specification and within the scope of Ada library packages CONDITION_HANDLING and SYSTEM. The use of the pragma does not forbid other Ada calls to the entry. The entry may have at most one formal parameter, of a discrete, address, or access type, with mode in only. The name must not denote an entry family or a member of an entry family. The pragma must be used in conjunction with the AST_ENTRY implementation-defined attribute in a call to the package Starlet QIO operation, or, alternatively the package TASKING_SERVICE, as provided by the Ada runtime library. The call must occur within the body of the associated task. The occurrence of an AST queues the call onto the entry (a scheduling point); the AST is then subsequently dismissed. Normal priority assignments and Ada tasking rules will determine the servicing of the AST entry call. In particular, the entry rendezvous does not execute "at AST level."

This pragma is not supported by the SD XD Ada cross-target compilers.

```
CALL_SEQUENCE_PROCEDURE
   ([ [UNIT =>] internal_name
    [, [RESULT_TYPE =>] type_mark ]
    [, [PARAMETER_TYPES =>] ( parameter_types ) ]
    [, [MECHANISM =>] mechanism ]
    [, [RESULT_MECHANISM =>] mechanism_spec]
    [, [PRESERVED_REGISTERS =>] ( registers ) ] )

CALL_SEQUENCE_FUNCTION
   ([ [UNIT =>] internal_name
    [, [RESULT_TYPE =>] type_mark ]
    [, [PARAMETER_TYPES =>] ( parameter_types ) ]
    [, [MECHANISM =>] mechanism ]
    [, [RESULT_MECHANISM =>] mechanism_spec]
    [, [PRESERVED_REGISTERS =>] ( registers ) ] )
```

Description: These directives specify interface characteristics for exported procedures and functions. These pragmas are identical to the EXPORT_PROCEDURE and EXPORT_FUNCTION pragmas described below.

The CALL_SEQUENCE_PROCEDURE *and* CALL_SEQUENCE_FUNCTION *pragmas are not supported by the DEC VAX/VMS host/target compiler.*

```
IMPORT_PROCEDURE ( internal_name [, external_designator]
                   [, pragma_specific_options ] )

IMPORT_VALUED_PROCEDURE ...
IMPORT_FUNCTION ...
IMPORT_EXCEPTION ...
IMPORT_OBJECT ...
EXPORT_PROCEDURE ...
EXPORT_FUNCTION ...
EXPORT_EXCEPTION ...
EXPORT_OBJECT ...
```

Description: These directives allow a wide variety of entities to be imported or exported. An external designator may be given as a string literal. External designators must be valid VAX/VMS or XD linker identifiers, but not necessarily valid Ada identifiers. Type checking is not performed across the interface and exported symbol resolution is deferred until link time. Pragma_specific_ options allow the specification of PARAMETER_TYPES, RESULT_TYPES, parameter passing MECHANISM, and result MECHANISM. MECHANISM refers to the method of passing parameters or returning results — value, reference, or descriptor. SD XD Ada allows registers to be specified for passing parameters, such as:

```
REFERENCE ( REGISTER => R3 )
```

The pragmas IMPORT_PROCEDURE and IMPORT_FUNCTION are used in conjunction with a previous occurrence of pragma INTERFACE for the subprogram. INTERFACE can be used alone and the import pragmas will be implicitly declared with default values. The SD XD Ada implementation also supports DOPE_VECTOR and BIT_DOPE_VECTOR parameter passing mechanisms plus IMPORT, EXPORT, and PRESERVED_REGISTERS as pragma_specific_options. IMPORT_VALUED_PROCEDURE allows a valued procedure to be called from an Ada program. The returned value typically represents an environment status.

SD XD Ada does not support the IMPORT_VALUED_PROCEDURE *pragma.*

LEVEL (integer_literal)

Description: Directive specifies the level (range 1..3) of interrupt association with a task. Gradations in level have differing mechanisms for handling the interrupt, from fast to slow.

This pragma is only supported by the SD XD Ada cross-target compiler.

**LINK_OPTION ([[[TARGET =>] link_options_file_name]
[, [MAPPING =>] link_options_file_name]])**

Description: Directive specifies an association of program to linker characteristics that must be given within the named link_options_file_name files. By placing these switches within the Ada program, the need to specify each characteristic is not required on the link command line. The link option files may be entered into the XD ACS library and modified without the need for program recompilation. The use of ACS option files reduces the need for retaining external script files outside of XD Ada version control.

LONG_FLOAT (D_FLOAT | G_FLOAT)

Description: Directive specifies the internal representation chosen for the predefined type LONG_FLOAT and for floating point type declarations with digits specified in the range 7..15. The default is G_FLOAT. The pragma must appear before the first compilation unit. Package SYSTEM, which declares the floating point types, must be visible at the place where the pragma appears. Use of pragma LONG_FLOAT implies a recompilation of the predefined STANDARD environment and all units dependent on the pragma. Based on the pragma's argument, VAX can offer one of four predefined floating-point representations with differing levels of numeric precision.

This pragma is not supported by the SD XD Ada cross-target compiler.

**PSECT_OBJECT (internal_name [, psect_designator]
[, [SIZE =>] size_designator])**

Description: Directive specifies the shared use of variables that are stored in overlayed linker program sections (psects). The pragma allows only one object to be allocated in a particular program section. The object must not have an initial value. The effect of storing multiple objects can be obtained by using a record object, with each record component corresponding to an external variable. The internal_name must be a valid Ada identifier. The psect_designator names the program section and can be either an Ada identifier or a string denoting a VAX/VMS name. The SIZE designator specifies a VAX/VMS Linker absolute global symbol that will be defined in the object module. SIZE may be used to achieve link-time consistency checking. DEC makes this operation available on all VAX-hosted compilers. The intent is obvious — a common and consistent approach to multi-language support, with a single linker interface.

This pragma is not supported by the SD XD Ada cross-target compiler.

SUPPRESS_ALL

Description: Directive specifies that all runtime checks in a compilation unit be suppressed. This pragma supersedes the language-defined SUPPRESS pragma.

TASK_STORAGE ([TASK_TYPE =>] simple_name,
[TOP_GUARD =>] static_simple_name)

Description: Directive specifies additional storage (guard pages) for each task activation. Guard pages provide protection against storage overflow during task execution of non-Ada code. During execution of non-Ada images, the exception STORAGE_ERROR cannot be detected. A stack overflow which crosses over a guard page, of possible zero size, will modify memory locations in an undetermined manner. Similarly, stack overflows which cross into a guard page of non-zero size will raise a hardware access violation (SS$_ACCVIO) exception, terminating the main image. Another VAX pragma, IMPORT_EXCEPTION, allows non-Ada exceptions to be handled in Ada programs. Another alternative is to declare NON_ADA_ERROR as an exception. When this exception is given as a choice in an Ada exception handler, it matches itself or any VAX/VMS exception; this allows non-Ada conditions to be treated as a special subclass of Ada exceptions.

This pragma is not supported by the SD XD Ada cross-target compiler.

TIME_SLICE (static_expression)

Description: Directive specifies a value of type DURATION to be used as a round-robin scheduling interval for execution of tasks with equal priority. This operation is provided by DEC as a VAX multi-language support feature.

This pragma is not supported by the SD XD Ada cross-target compiler.

TITLE (titling_option [, titling_option])

```
titling_option ::= [ TITLE => ] string_literal |
                   [ SUBTITLE => ] string_literal
```

Description: Directive specifies a new title or subtitle string at the top of a compilation listing. This supersedes the default title and/or subtitle. This operation is provided by DEC as a VAX multi-language support feature.

VOLATILE (variable_simple_name)

Description: Directive specifies that all references and updates to the named object (of any type) are asynchronous; no local copies of the variable are allowed. The pragma must appear following the object declaration within the same declarative region and preceding the first named occurrence of the variable, other than in an address clause. VOLATILE replaces the language-defined pragma SHARED. The use of VOLATILE is not limited to scalar or access types, as is SHARED.

Implementation-Defined Attributes

P'AST_ENTRY

Description: Given a prefix P which denotes the entry of a task object, AST_ENTRY yields a value of type AST_HANDLER (defined in package SYSTEM) that transforms an AST occurrence into a call of the given entry. The attribute is used in conjunction with pragma AST_ENTRY for the purpose of handling VAX/VMS system services.

This attribute is not supported by the SD XD Ada cross-target compiler.

P'BIT

Description: Given a prefix P that denotes an object, BIT yields the bit offset within the storage unit containing the first bit of storage for the object P. BIT returns a value of type *universal_ integer.*

P'MACHINE_SIZE

Description: Given a prefix P that denotes any type or subtype, MACHINE_SIZE yields the actual number of bits to be allocated for variables of type P, including padding across unit boundaries. MACHINE_SIZE returns a value of type *universal_integer.*

P'NULL_PARAMETER

Description: Given a prefix P that denotes any type or subtype, NULL_PARAMETER yields a dummy object of type P allocated at machine address zero. NULL_PARAMETER is used as a default expression on formal parameters of VAX/VMS system service calls or as an actual expression on such a call. The called subprogram must be imported using the pragma IMPORT.

This attribute is not supported by the SD XD Ada cross-target compiler.

P'TYPE_CLASS

Description: Given a prefix P that denotes a type or subtype, TYPE_CLASS yields the value of the type class for the "full" type of P. TYPE_CLASS returns a value of the type TYPE_CLASS as declared in package SYSTEM. The TYPE_CLASS attribute operation gives the user the ability to determine the underlying implementation type of which P is a member. This anonymous type may be one of nine VAX entity representation classes, as follows:

```
type TYPE_CLASS is (TYPE_CLASS_ENUMERATION,
                    TYPE_CLASS_INTEGER,
                    TYPE_CLASS_FIXED_POINT,
                    TYPE_CLASS_FLOATING_POINT,
                    TYPE_CLASS_ARRAY,
                    TYPE_CLASS_RECORD,
                    TYPE_CLASS_ACCESS,
                    TYPE_CLASS_TASK,
                    TYPE_CLASS_ADDRESS);
```

This information can be used for selecting a suitable object descriptor when specifying a parameter passing mechanism in an import or export pragma.

4.5. Harris Corporation

Observed in the following compilation systems, except when noted:

Harris Ada Compiler Version 4.0
Harris H1200, VOS Version 7.1 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Harris Ada Compiler Version 4.0
Harris HCX-9, HCX/UX Version 3.0 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **CONTROLLED**
- **MEMORY_SIZE**
- **OPTIMIZE**
- **PRIORITY** *No information provided.*
- **SHARED** *No information provided.*
- **SUPPRESS** *Modified — check suppression on named objects is not supported.*
- **STORAGE_UNIT**
- **SYSTEM_NAME**

Implementation-Defined Pragmas

EXTERNAL_NAME (simple_name, string_literal)

Description: Directive associates a declared object or subprogram with an external entity, whose name is given by the string literal. The external name is not required to be a valid Ada identifier. Similar to DDC, Tandem, and Verdix pragmas. This pragma is used to export Ada program entities used by external programs.

INTERFACE_OBJECT (simple_name [, string_literal])

Description: Directive associates a declared object with an external object whose name is optionally given by the string literal. The default is the Ada name converted to lower case. Similar to the DEC VAX Ada pragma IMPORT, EXPORT, PSECT_OBJECT, Rational IMPORT_OBJECT and Verdix INTERFACE_OBJECT.

INTERFACE_COMMON_OBJECT (simple_name, string_literal)

Description: Directive associates a declared object with an externally defined "common block" identified by the string literal. Similar to the previous pragma except that the second parameter is not optional.

This pragma is not supported in HCX-9 host/targeted compiler.

INTERFACE_MCOM_OBJECT (simple_name, string_literal, string_literal)

Description: Directive associates the given Ada name with a foreign object identified to the linker by the second string literal and located in a disk sector identified by the third string literal as a "Monitor Common" area. No further details provided in Appendix F.

This pragma is not supported in HCX-9 host/targeted compiler.

SHARED_PACKAGE

Description: Directive specifies that the library package specification immediately following this pragma be considered a *shared* entity. The pragma also applies to nested package specifications within the library package. Objects within the shared library package specification must not be initialized, must not be of an access type, and must not be unconstrained.

This pragma is not supported in HCX-9 host/targeted compiler.

SHARE_BODY (generic_name, *boolean*_literal)

Description: Directive specifies that the named generic unit or instantiated unit may share a single instantiated body when the second parameter is TRUE. The default is for sharing of instantiated units except in the presence of pragma INLINE. Other usage restrictions apply.

IMPLICIT_CODE ([ON | OFF])

Description: Directive specifies to the compiler that implicit code is allowed (ON) or not allowed (OFF). This pragma is restricted to the declarative part of a machine code procedure. The program must make use of the system package MACHINE_CODE for this purpose. Implicit code is that code generated by the runtime for managing the return address and parameters.

This pragma is not supported by the Harris H1200 host/target compiler.

OPT_LEVEL ([NONE] | [MINIMAL] | [GLOBAL] | [MAXIMAL])

Description: Directive specifies the level of compiler optimization to be performed. The default level is MINIMAL.

This pragma not supported in HCX-9 host/targeted compiler.

Implementation-Defined Attributes

P'LOCK (boolean_expression [, integer_expression])

P'UNLOCK

Description: Returns a boolean value representing the LOCKed (FALSE) or UNLOCKed (TRUE) status of the package. The package known by the prefix P must have been named previously in a pragma SHARED_PACKAGE. The attribute LOCK performs a test and set operation on the designated package. The requester is queued on a semaphore if the package is currently in a LOCKed state. The program waits for the semaphore to be unlocked if the first parameter's value is TRUE. The value FALSE indicates a no-wait request (as in polling). The second parameter gives the number of clock ticks prior to wake-up and awaiting a TRUE status. The second parameter defaults to zero. The UNLOCK attribute alters the state of the package to UNLOCKed and returns a boolean value indicating the current state as given above. Neither attribute protects the SHARED_PACKAGE from concurrent access other than through the implicit semaphore operations provided via LOCK and UNLOCK.

These attributes are not supported in HCX-9 host/targeted compiler.

4.6. Intermetrics, Inc.

Observed in the following compilation systems,
except when noted:

MVS Ada Real-Time Compiler Version 202.35
IBM 4341, MVS/370 1.3.4 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

UTS Ada Real-Time Compiler Version 202.35
IBM 3083 (S/370), UTS 2.3 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

CMS Ada Real-Time Compiler Version 202.35
IBM 3083, CMS 4 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **MEMORY_SIZE**
- **OPTIMIZE**
- **SHARED**
- **STORAGE_UNIT**
- **SUPPRESS** *Modified — check suppression on named objects is not supported.*
- **SYSTEM_NAME**

Implementation-Defined Pragmas

SUPPRESS_ALL

Description: Directive suppresses all runtime checks for the compilation unit affected. Similar in effect to the pragma of the same name from Concurrent Computer, DEC, and System-Designers.

Implementation-Defined Attributes

No implementation-defined attributes are defined by this vendor.

4.7. Irvine Compiler Corp.

Observed in the following compilation systems,
except when noted:

ISI Optimum V, ISI 4.3 BSD Release 4.0D host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

ICC Ada Version 6.0 Release 5.0
Hewlett-Packard, Model 9000/350 HP-UX host/target,
Validation Summary Report, Appendix B (Ada RM Appendix F)

ICC Ada Version 2.0 Release 5.0
Hewlett-Packard, Model 9000/825 HP-UX host/target,
Validation Summary Report, Appendix B (Ada RM Appendix F)

DEC MicroVAX 2000 VMS Version 4.7 host,
Intel i80960KB SBC, bare machine target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **LIST**
- **MEMORY_SIZE**
- **STORAGE_UNIT**
- **SYSTEM_NAME**

The language-defined pragma SUPPRESS has been modified to allow the following names and effects:

- ***Exception_Info*** — a runtime optimization which causes a reduction of allocated storage for messages generated when exceptions are propagated out of the main program or a task.
- ***All_Checks*** — suppresses all standard checks plus Exception_Info.

Implementation-Defined Pragmas

COMPRESS (discrete_subtype_name)

Description: Directive specifies a reduction in storage for the named subtype when objects of the subtype are components of an array or record. The compaction is independent of the representation of the base type. The pragma must appear prior to any reference to the named subtype. This is a performance enhancement feature aimed at space optimization.

EXPORT (foreign_language_name, Ada_subprogram_name [, linker_name_string])

Description: Directive specifies that the Ada subprogram be made visible to foreign calling units of another language. The pragma must appear prior to the body of the designated unit. Identical to TLD pragma EXPORT.

NO_ZERO (record_subtype_name)

Descriptions: Directive suppresses zero-initialization of unused bits in a record. If the record has all fields adjacent then the the pragma has no effect. When zeroing of record gaps is disabled, comparisons of the named type are disallowed. The use of this pragma can reduce initialization time for record objects. This pragma is a performance enhancement feature aimed at time optimization.

PUT (static_expression | string_literal {, static_expression | string_literal })
PUT_LINE (static_expression | string_literal {, static_expression | string_literal })

Description: The specified values are written to the standard output at compile-time. Pragma PUT_LINE adds a carriage return following last output argument value. These pragmas are used in conjunction with conditional compilation commands.

IF (compile_time_boolean_expression)
ELSIF (compile_time_boolean_expression)
ELSE
END

Description: Structured commands, akin to the Ada if-statement, that control conditional compilation of text. A preprocessor pass is required for these pragmas.

INCLUDE (file_path_name_string)

Description: Directive specifies inclusion of a source file at the point where the pragma appears. The pragma is known to the library manager for purposes of closure and recompilation. This pragma requires a compiler preprocessor pass.

Implementation-Defined Attributes

P'VERSION
P'SYSTEM
P'TARGET

Description: Return a value dependent upon the target architecture and operating system. This attribute is used for conditional compilation. The prefix must be a discrete type. No further details are available from Appendix F.

4.8. Meridian Software Systems, Inc.

Observed in the following cross-compilation systems, except when noted:

Meridian AdaVantage Version 2.2
ALR 386/2 IBM PC DOS 3.3 host/target,
Validation Summary Report, Appendix B (Ada RM Appendix F)

Meridian AdaVantage Version 2.2
Apple Macintosh Plus System, Finder 6.0, System 4.2 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Meridian AdaVantage Version 2.2
Apple Macintosh II System 4.2 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Meridian AdaVantage Version 2.2
IBM PS/2 Model 60 with Floating Point Co-Processor, IBM PC DOS 3.3O host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

IBM PS/2 Model 30 with Floating Point Co-Processor, IBM PC DOS 3.3O host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Meridian AdaVantage Version 2.2
ITT XTRA/286 with Floating Point Co-Processor, MS/DOS 3.2 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Meridian AdaVantage Version 2.2
Zenith Z-248 with Floating Point Co-Processor, MS/DOS 3.10 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Meridian AdaVantage Version 2.2
Zilog System 8000, Zeus 3.21 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **MEMORY_SIZE**
- **STORAGE_UNIT**
- **SYSTEM_NAME**

Implementation-Defined Pragmas

No implementation-defined pragmas are provided by this vendor.

Implementation-Defined Attributes

No implementation-defined attributes are provided by this vendor.

4.9. Rational, Inc.

*Observed in the following cross-compilation systems,
except when noted:*

*Rational Environment Version 3.0 (Delta)
R1000 host/target
Appendix F*

*Rational Environment Version D_10_8_6
R1000 Series 200 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)*

*Rational VAX_VMS Version 2.0.45
R1000, D_10_9_10wps host
DEC VAX 11/750, VMS Version 4.5 target
Validation Summary Report, Appendix B (Ada RM Appendix F)*

*Rational MC689020/OS-2000 Version 2.0.30
R1000, D_10_9_10wps host
Phillips PG2100, OS-2000 target
Validation Summary Report, Appendix B (Ada RM Appendix F)*

*Rational MC68020_Bare Version 2.0.30
R1000, D_10_9_10wps host
Motorola 68020 MVME 135, bare machine target
Validation Summary Report, Appendix B (Ada RM Appendix F)*

*Rational M1750a Version 2.0.122
R1000, D_10_9_1_1750a host
Mikros MKS1750/SO, bare machine
Validation Summary Report, Appendix B (Ada RM Appendix F)*

Language-Defined Pragmas

Appendix F indicates that all language-defined pragmas are supported.

Implementation-Defined Pragmas

DISABLE_DEALLOCATION (T)

Description: Directive disables automatic storage reclamation for objects of type T. This pragma can be used to nullify the effect of UNCHECKED_DEALLOCATION (Ada RM 13.10.1) on objects of type T.

This pragma is not supported by Rational cross-compilers.

ENABLE_DEALLOCATION (T)

Description: Directive enables automatic storage reclamation for type T. The pragma is used with UNCHECKED_DEALLOCATION to enable UNCHECKED DEALLOCATION for objects of type T. The pragma may be used to indicate that a generic formal type is deallocatable. This pragma, as with DISABLE_DEALLOCATION, is only meaningful in conjunction with UNCHECKED_DEALLOCATION.

This pragma is not supported by Rational cross-compilers.

LOADED_MAIN

Description: Directive specifies to the Rational Environment that a (program) unit is *coded* only. The effect is that the archiving librarian maintains the unit in a code-generated state. The absolute load image has not yet been formed. This is the equivalent of a VAX Sharable Image. Rational program units exist in one of four states of readiness (from lowest to highest, respectively):

1. archived — storage-encrypted image
2. source — compilable source program
3. installed — semanticized program with DIANA tree (intermediate form) created
4. coded — relocatable objects generated

This pragma is not supported by Rational cross-compilers.

MAIN

Description: Directive specifies that the Rational Environment is to preload any compilation units referenced by a main program. The pragma must appear immediately after the declaration of the main subprogram. The effect takes place only after the main subprogram has been promoted to a coded state, and all units referenced by the main subprogram have also been promoted to the coded state. Following closure, a "load-and-go" command will automatically appear in the Command window referencing these units.

The Rational cross-target compilers (1750a/68020) version 5.0 have modified forms of this pragma with an additional formal part as follows:

```
MAIN ( [ TARGET     => target_name ]
       [, STACK_SIZE => static_integer_expression ]
       [, HEAP_SIZE  => static_integer_expression ] )
```

NICKNAME (string_literal)

Description: Directive specifies a unique name for an Ada subprogram when importing and exporting overloaded subprograms.

This pragma is not supported by the Rational Environment R1000 host/target compiler.

MUST_BE_CONSTRAINED ([conditional_expression =>] type_id, ...)

Description: Directive specifies that a generic formal limited or private type must be constrained. The pragma provides the Rational environment with the ability to check for legal instantiations prior to installation of the generic body.

This pragma is not supported on Rational cross-compilers.

OPEN_PRIVATE_PART

Description: Directive specifies that a Rational Subsystem interface has an open private part. *Subsystems* are analogous to Ada package specifications and are divided into a public and private part.

This pragma is not supported on Rational cross-compilers.

PAGE_LIMIT (X)

Description: Directive specifies that the memory page limit for the current job should be no less than X, where X is a whole number.

This pragma is not supported on the Rational M1750a compiler.

PRIVATE_EYES_ONLY

Description: Directive specifies that after occurrence of the pragma, the units named in context clauses are needed only in the private part of a Subsystem interface.

This pragma is not supported on the Rational M1750a compiler.

```
IMPORT_FUNCTION ( [ INTERNAL => ] internal_name
                  [, [ EXTERNAL => ] external_name ]
                  [ [, [ PARAMETER_TYPES => ] parameter_types ]
                  [, [ RESULT_TYPE => ] type_mark ] |
                  [, [ NICKNAME => ] string_literal ] ]
                  [, [ MECHANISM => ] VALUE | REFERENCE ] )
```

IMPORT_PROCEDURE ...
EXPORT_PROCEDURE ...
EXPORT_FUNCTION ...
IMPORT_OBJECT ...
EXPORT_OBJECT ...

Description: Directives specify a mapping between Ada entities and external entities. This pragma is similar to DEC VAX Ada pragma IMPORT, EXPORT, PSECT_OBJECT, Harris INTERFACE_OBJECT and Verdix INTERFACE_OBJECT.

These pragmas are not supported by the Rational Environment R1000 host/target compiler.

Implementation-Defined Attributes

No implementation-defined attributes are provided by this vendor.

4.10. R. R. Software, Inc.

Observed in the following compilation systems, except when noted:

JANUS/Ada Version 2.0.2
IBM PS/2, Model 80, IBM PC-DOS 3.30 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

IntegrAda Version 2.0.1
IBM PS/2, Model 80, IBM PC-DOS 3.30 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

IntegrAda Version 2.0.1
Zenith Z-184, Zenith MS-DOS 3.21 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

IntegrAda Version 2.0.1
Zenith Z-248, Zenith MS-DOS 3.21 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

IntegrAda Version 2.0.1
PC's Limited 386, Compaq MS-DOS 3.21 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

JANUS/Ada Version 2.0.2
Zenith Z-183, Zenith MS-DOS 3.30 host/target
Validation Summary Report, Appendix B (Ada PM Appendix F)

JANUS/Ada Version 2.0.2
Zenith Z-284, Zenith MS-DOS 3.21 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

JANUS/Ada Version 2.0.0
Compaq Deskpro 286, MS-DOS 3.1 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

JANUS/Ada Version 2.0.0
IBM PC/XT, IBM PC-DOS 3.3 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

JANUS/Ada Version 2.0.0
NCR Worksaver 300, MS-DOS 2.1 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

JANUS/Ada Version 2.0.0
PC's Limited 386, Compaq's MS-DOS 3.1 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **CONTROLLED**
- **INLINE**
- **INTERFACE**
- **MEMORY_SIZE**
- **PACK**
- **PRIORITY**
- **STORAGE_UNIT**
- **SUPPRESS** *Modified to support a single argument only.*
- **SYSTEM_NAME**

Implementation-Defined Pragmas

ALL_CHECKS ([ON] | [OFF])

Description: When OFF is specified, all exception checks are suppressed. Checks ON is the default. The effect of the pragma is undefined when used in combination with pragma SUPPRESS.

ARITHCHECK ([ON] | [OFF])

Description: When OFF is specified, DIVISION_CHECK and OVERFLOW_CHECK are suppressed. Checks ON is the default. The effect of the pragma is undefined when used in combination with pragma SUPPRESS.

CLEANUP (Integer_literal)

Description: Directive specifies the level of automatic storage reclamation for stack space recovery. Level 3 is the default level for normal recovery of all stack space; level 0 implies no recovery whatsoever. This pragma is a performance enhancement feature aimed at time optimization. Incorrect use of the pragma may lead to STORAGE_ERROR.

DEBUG ([ON] | [OFF])

Description: Directive causes the generation of line numbers and procedure name runtime code for tracing unhandled exceptions. The default is ON. This pragma is a performance enhancement feature aimed at space optimization, with the cost being low error traceability.

ENUMTAB ([ON] | [OFF])

Description: When OFF is specified, tables supporting the IMAGE, VALUE, and WIDTH attributes are not generated for enumeration types. The default is ON. This pragma is a performance enhancement feature aimed at space optimization with the cost being the inability to use the IMAGE, VALUE, and WIDTH attributes for enumeration types, and inability to use ENUMERATION_IO.

INCLUDE (string_literal)

Description: Directive specifies the name of an MS-DOS Ada program file to be included at the place of the pragma. This operation is similar to the VAX Pascal INCLUDE command.

PAGE_LENGTH (integer_literal)

Description: Directive specifies the number of source lines per page for listings.

RANGECHECK ([ON] | [OFF])

Description: When OFF, RANGE_CHECK is suppressed. The default is ON. The effect of this pragma is undefined when used in combination with pragma SUPPRESS.

RESTRICTED_ENUMERATIONS

Description: When OFF is specified, tables supporting the IMAGE, VALUE, and WIDTH attributes are not generated for enumeration types. The default is ON. This pragma is a performance enhancement feature aimed at space optimization with the cost being the inability to use the IMAGE, VALUE, and WIDTH attributes for enumeration types, and inability to use ENUMERATION_IO. The difference in effect, if any, between RESTRICTED_ENUMERATIONS and ENUMTAB, other than the span of control, is not clear from the vendor's Appendix F.

RESTRICTED_OPERATORS (integer_literal)

Description: Directive specifies to the compiler that an operator associated with the number given in the parameter as a value in the range 0..100 will not be used in the application. This pragma is for the internal use of R.R. Software Inc. for debugging purposes. No further information is offered in the vendor's Appendix F with regard to which operators are associated with which numbers, the effect of the use of the pragma, or any other aspects resulting from its use.

RESTRICTED_RECORDS

Description: Directive specifies to the compiler that the application will not derive record types nor use record aggregates. The compiler will not generate the necessary data structures for these activities. This pragma is a performance enhancement feature aimed at space optimization.

RESTRICTED_SUBPROGRAMS

Description: Directive specifies to the compiler that the application will not derive user-defined types or make use of function calls in default expressions for parameters. This pragma is a performance enhancement feature aimed at space optimization. For all the RESTRICTED_.. pragmas, the vendor leaves undocumented (in Appendix F) the effect of violating any of the restrictions.

SYSLIB (integer_literal)

Description: Directive specifies to the compiler that the compilation unit is a member of a recognized (JANUS/Ada) system library. The parameter must be in the range of 1..15. The vendor indicates that the use of this pragma should be restricted to those cases where the user is writing replacement libraries for the runtime system.

VERBOSE ([ON] | [OFF])

Description: Directive specifies the extent of error information provided by the compiler in the error listing. When specified ON, the scope of the error extends to the two lines preceding the error line with an arrow pointing at the error line itself. When OFF is specified, only the error line number is provided.

Implementation-Defined Attributes

No implementation-defined attributes are provided by this vendor.

4.11. System Designers, Inc.

Observed in the following compilation systems,
except when noted:

SD Ada-Plus, Version 3B.01
DEC VAX/VMS host,
Motorola MC68020, bare machine target
Appendix F

Note: Refer to Digital Equipment Corporation for details
on the XD Ada Compiler for VAX/VMS to Motorola MC680X0

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **CONTROLLED**
- **INLINE**
- **MEMORY_SIZE**
- **OPTIMIZE**
- **SHARED**
- **STORAGE_UNIT**
- **SUPPRESS**
- **SYSTEM_NAME**

Implementation-Defined Pragmas

DEBUG ([NAME =>] expression)

Description: Directive specifies the generation of "out-of-line" code that writes the argument value to a buffer. The code is optionally executed by the debugger when a breakpoint is inserted at the position of the pragma in the code. The pragma may appear anywhere a declaration or a statement may appear.

EXPORT ([ADA_NAME =>] name, [EXT_NAME =>] string_literal)

Description: Directive associates an external name, specified with a string literal, with the name of a data object. The string literal must provide a valid name for an object in MC68020 Assembler. The pragma may appear as a basic declarative item of library package specification or in the declarative part of a library package body. Pragma EXPORT is similar in intent to the Alsys INTERFACE_NAME, the Intel LINKNAME, and the DDC INTERFACE_SPELLING pragmas. DEC subsumes this feature within their overall IMPORT/EXPORT pragma facility.

SQUEEZE (type_simple_name)

Description: Directive specifies bit level storage minimization for any objects of the named type, which must be a record type or an array type. The pragma must appear before the first occurrence of an object of this type. Pragma SQUEEZE is provided in addition to the language-defined pragma PACK. PACK is implemented to allocate storage down to the byte level, with gaps between record components reduced to a fraction of a byte.

SUPPRESS_ALL

Description: Directive specifies the runtime suppression of checks for CONSTRAINT_ERROR and NUMERIC_ERROR. The pragma must appear before the first non-comment line in a program unit. The language-defined pragma SUPPRESS is not supported. There is no mention of suppressing checks for TASKING_ERROR or PROGRAM_ERROR.

SUPPRESS_STACK

Description: Directive specifies the runtime suppression of checks for STORAGE_ERROR.

Implementation-Defined Attributes

No implementation-defined attributes are provided by this vendor.

4.12. Tandem Computers

Observed in the following compilation systems, except when noted:

Tandem Ada Version T9270C10
Tandem NonStop VLX, GUARDIAN 90 Version C10 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **MEMORY_SIZE**
- **OPTIMIZE**
- **PACK**
- **STORAGE_UNIT**
- **SUPPRESS**
- **SYSTEM_NAME**

Implementation-Defined Pragmas

CONDITION_CODE (subprogram_name)

Description: When the named subprogram has been associated with an external subprogram (by means of pragma INTERFACE), this directive specifies an operating system status or condition code will be returned when the Ada program is called. This pragma can only be used in conjunction with a special runtime package that exports a function returning a condition code.

EXTENSIBLE_ARGUMENT_LIST (subprogram_name)

Description: When the named subprogram has been associated with an external subprogram (by means of pragma INTERFACE), this directive specifies that the subprogram can be called with an "extensible argument list." No further description of the pragma is made available in the vendor's Appendix F, or an explanation of the meaning of "extensible argument list."

EXTERNAL_NAME (subprogram_name, string_literal)

Description: When the named subprogram has been associated with an external subprogram (by means of pragma INTERFACE), this directive specifies a different external name.

PRIMARY (variable_name { , variable_name } ...)

Description: Directive specifies the allocation of primary memory for the named objects rather than the default use of extended memory storage. Other restrictions apply to the use of the pragma. The references to primary and extended memory are not related to static, heap, or stack object allocation.

VARIABLE_ARGUMENT_LIST (subprogram_name)

Description: When the named subprogram has been associated with an external subprogram (by means of pragma INTERFACE), this directive specifies that the named subprogram has a "variable argument list". No further description of the pragma is made available in the vendor's Appendix F, or an explanation of the meaning of "variable argument list."

Implementation-Defined Attributes

P'EXTENDED_ADDR

Description: Returns the 32-bit address of a prefix P, which must be an Ada variable declared in a visible object declaration. This pragma is identical in all respects to the supported P'ADDRESS language-defined attribute.

P'WORD_ADDR

Description: Returns the 16-bit address of a prefix P, which must be an Ada variable declared in a visible object declaration. The variable must reside in primary memory (lower 64K); otherwise, CONSTRAINT_ERROR will be raised.

P'STRING_ADDR

Description: Returns the 16-bit string address of a prefix P, which must be an Ada string variable declared in a visible object declaration. The string variable must reside in primary memory (lower 64K); otherwise, CONSTRAINT_ERROR will be raised.

4.13. Tartan Laboratories

Observed in the following compilation systems,
except when noted:

Tartan Ada VMS 1750A Version 2.0,
DEC VAX/VMS host,
MIL-STD 1750A, bare machine target
Appendix F

Tartan Ada VMS 68K Version 0.3 Beta
DEC VAX/VMS host,
Motorola MC680X0, bare machine target
Appendix F

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **INTERFACE** *Replaced by* FOREIGN_BODY *below.*
- **LIST** *Replaced by a multi-level compiler option switch on the 1750A cross-compiler and supported with restrictions (compile switch LIST=ALWAYS) on the MC680X0 cross-compiler*
- **OPTIMIZE** *Replaced by multi-level compiler option switch.*
- **PAGE** *Replaced by embedding formfeed control characters in source text on the 1750A cross-compiler and supported with restrictions (compile switch LIST=ALWAYS) on the MC680X0 cross-compiler.*
- **SUPPRESS** *Replaced by compiler option switch equivalent to an ALL CHECKS suppression on the 1750A cross-compiler only; pragma is fully supported on MC680X0 cross-compiler.*

A warning message is provided if any unsupported language-defined pragma is found in the program source text.

Implementation-Defined Pragmas

LINKAGE_NAME (simple_name, string_literal)

Description: Directive associates an external name with the simple name. The string literal need not be a legal Ada name. The pragma must appear after the declaration of the named entity within a package specification. The pragma appears to be restricted to exported entities. See also notes on the SDS EXPORT pragma.

FOREIGN_BODY (language_name [, elaboration_routine_name])

Description: Directive specifies a non-Ada subprogram as callable from the Ada program unit. The subprogram must be in the form of an object module which represents the body of a non-generic library package. The bodies for several subprograms may be contained in one object module. The language name identifies the calling convention for the foreign module. Restrictions apply to the module such that the above conventions and data representations are compatible with the Tartan Ada compiler. Subprograms called by tasks must be reentrant. The optional elaboration routine name provides a mechanism for identifying a global routine in the object module to be executed first in order to initialize any necessary data objects. The package specification which uses this pragma must contain only subprogram declarations, objects declared with an unconstrained type mark, and named numbers. Type marks may not denote task types and objects must not be given initial values. The pragma must appear before the first declaration in the package specification. An Ada body may substitute for the foreign body, regardless of the language specified, by either compiling it directly into the library or using a Tartan command to reference another Ada library for the Ada body. The pragma LINKAGE_NAME must be used in all the above cases for each exported entity.

Implementation-Defined Attributes

No implementation-defined attributes are provided by this vendor.

4.14. Telesoft, Inc.

Observed in the following compilation systems,
except when noted:

Telegen2 Ada Development System Version 3.23
DEC MicroVAX II, VMS 4.7 host,
Motorola MC680X0 MVME-133 with MC68881 Floating-Point
Co-processor, bare machine target
Appendix F

Telegen2 Ada Development System Version 3.22a
DEC MicroVAX II, VMS 4.7 host,
Motorola MC680X0 MVME-133 with MC68881 Floating-Point
Co-processor, bare machine target
Appendix F

Telegen2 Ada Development System Version 3.22
DEC MicroVAX II, VMS Version 4.6 host
ECSPO RAID Simulator MIL-STD 1750A Version 4.0, bare machine target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Telegen2 Ada Development System Version 3.20
DEC MicroVAX II, VMS Version 4.6 host
Motorola MC68020 MVME-133A-20 with MC68881 Floating-Point
Co-processor, bare machine target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Telegen2 Ada Development System Version 3.20
DEC MicroVAX II, VMS Version 4.6 host
Motorola MC68010 MVME-117-4, bare machine target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Telegen2 Ada Development System Version 3.20
DEC MicroVAX II, VMS Version 4.6 host
Motorola MC68000 MVME-101, bare machine target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Telegen2 Ada Development System Version 3.20
DEC MicroVAX II, VMS Version 4.6 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Cray Ada Compiler 1.0
CRAY X-MP, UNICOS 3.0 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Cray Ada Compiler 1.0
CRAY 2, UNICOS 4.0 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

IBM Development System for the Ada Language System (DS/ALS), Version 2.1.0
4381 VM/HPO Release 4.2 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

IBM Development System for the Ada Language System (DS/ALS), Version 2.1.0

IBM 4381 VM/HPO Release 4.2 host
IBM 4381 MVS/XA Release 2.1.7 target
Validation Summary Report, Appendix B (Ada RM Appendix F)

IBM Development System for the Ada Language System (DS/ALS), Version 2.1.0
IBM 4381 MVS/XA Release 2.1.7 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

IBM Development System for the Ada Language, VM/CMS -> AIX/RT Ada Cross-Compiler, Version 1.1.0
IBM 3083, VM/HPO Release 4.2 host
IBM RT PC 6150-125, AIX Release 2.1 target
Validation Summary Report, Appendix B (Ada RM Appendix F)

IBM Development System for the Ada Language, AIX/RT Ada Compiler, Version 1.1.0
IBM RT PC 6150-125, AIX Release 2.1 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Intel Inc., Ada-386 Version 3.20
DEC MicroVAX II, VMS Version 4.7 host,
Intel SBC i386, bare machine target.
RM Annotations Section C and Section 6.12.1.6.2
(Function_Mapping Optimizations), Appendix F

Gould Inc., APLEX Ada Compiler, Revision 2.1
Gould PowerNode Model 9080 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Gould Inc., APLEX Ada Compiler, Revision 2.1
NPI Model 4050 UTX/32 Revision 3.0 host
CONCEPT/32 Model 6780 MPX-32 Revision 3.4 target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Gould Inc. APLEX Ada Compiler, Revision 2.1
Gould PowerNode Model 9080, UTX/32 Revision 2.1 host
Gould CONCEPT/32 Model 6744, bare machine target
Derived host/target configurations:
Model 90XX/67XX, 60XX/67XX, 90XX/97XX
Validation Summary Report, Appendix B (Ada RM Appendix F)

Gould Inc., APLEX Ada Compiler Version 2.0
Gould CONCEPT/32 Model 9780, MPX-32 Version 3.4 host/target
Derived host/target configurations:
Model 97XX/97XX, 67XX/67XX, 97XX/67XX
Validation Summary Report, Appendix B (Ada RM Appendix F)

Gould Inc. APLEX Ada Compiler Version 2.0
Gould NP1 Model 4050, UTX/32 Version 3.0 host/target
Derived host/target configurations:
Model 40XX/40XX
Validation Summary Report, Appendix B (Ada RM Appendix F)

Gould Inc. APLEX Ada Compiler Version 2.0
Gould PowerNode Model 9080, UTX/32 Version 2.1 host/target

Derived host/target configurations:
90XX/90XX, 60XX/60XX, 90XX/60XX
Validation Summary Report, Appendix B (Ada RM Appendix F)

Macintosh Telegen2 Ada, Version 1.2
Macintosh II, Apple A/UX UNIX Release 1.0 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Sun Ada Version 1.2
Sun Microsystems Sun-3/280 Workstation, Sun UNIX Version 4.2
Release 3.2 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Sun Ada Version 1.2
Sun Microsystems Sun-3/280 Workstation, Sun UNIX Version 4.2
Release 3.2 host
Motorola MC68010 MVME 117-4, bare machine target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Sun Ada Version 1.2
Sun Microsystems Sun-3/280 Workstation, Sun UNIX Version 4.2
Release 3.2 host
Motorola MC68020 MVME 133A-20 with MC68881 floating-point
Co-processor, bare machine target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Sun Ada Version 1.2
Sun Microsystems Sun-3/280 Workstation, Sun UNIX Version 4.2
Release 3.2 host
Motorola MC68000 MVME-101, bare machine target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Sun Ada Version 1.3
Sun Microsystems Sun-4 Workstation (SPARC Processor),
Sun UNIX Version 4.2 Release Sys4-3.2 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

UNISYS Telegen2 Ada Compiler Version 3.20
UNISYS 5000/80/90, UNIX System V, Release 4.0 (1R1)
Motorola MC68020 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Language-Defined Pragmas

The following language-defined pragmas are not supported by the IBM DS/ALS compilers:

- **LIST**
- **MEMORY_SIZE**
- **PAGE**
- **STORAGE_UNIT**
- **SYSTEM_NAME**

Implementation-Defined Pragmas

COMMENT (string_literal)

Description: Directive specifies comments to be embedded into the object code image. This pragma is not allowed within generic formal parts.

This pragma is not supported by the IBM DS/ALS compiler.

IMAGES (enumeration_type [, DEFERRED | IMMEDIATE])

Description: Directive specifies the creation and allocation of an image table for the enumeration type. The default is IMMEDIATE, which generates the table following the declaration of the type. DEFERRED is an optimization which delays table allocation until an IMAGE, VALUE, or WIDTH attribute for the type is encountered. The effect of the optimization is best attained when a single compilation unit uses these attributes; the allocation occurs only once. Multiple unit references require individual table allocations for each unit — equivalent to an IMMEDIATE specification.

This pragma is not supported in the Intel cross-compiler, the IBM DS/ALS compiler, the Macintosh Telegen2 Ada compiler, the Ada Telegen2 compilers Version 3.20, the Sun Ada compilers, or the Cray Ada compiler.

LINKNAME (subprogram_name, string_literal)

Description: Directive associates an external name with an Ada subprogram. The pragma is used in conjunction with and immediately following a pragma INTERFACE. This is identical to Alsys' pragma INTERFACE_NAME except that Intel allows no intervening declarations.

This pragma is not supported by the IBM DS/ALS compiler, IBM VM/CMS->AIX compiler or IBM AIX compiler.

INTERRUPT (Function_Mapping)

Description: Directive designates a task as an interrupt handler. This directive eliminates a context switch when an interrupt occurs, but limits the body of the task to:

- an accept body alone,
- an accept body within a loop, or
- an accept body alternative within a selective wait.

The body of the accept statement may only reference statically allocated global data and the body may not interact with other tasks. The "mapping" is implemented as a function call within the designated task, rather than a procedure, in order to return the value of a possible loop expression. The function value returned is used to determine whether another loop is indicated. In this manner, a device status may be used to control the continued execution of the handler body. The pragma's argument, Function_Mapping, is a pseudo-parameter because it is the only accepted identifier.

```
task Controller is
begin
  entry Next_Command;
end Controller;

task body Controller is
begin
  pragma INTERRUPT (Function_Mapping);
  while not QIO_FIFO.Is_Empty loop
    accept Next_Command do
      -- work
    end Next_Command;
  end loop;
  ...
end Controller;
```

One interesting aspect of this mechanism is the ability to embed the interrupt service routine within a variety of selective wait profiles. All hardware entry calls are treated as conditional. Compare this pragma with that of INTERRUPT_HANDLER and PASSIVE in the Index.

This pragma is not supported in Telegen2 and APLEX cross-compiler, Telegen2 Ada compilers Version 3.20 for MC68000 target, IBM DS/ALS compiler, or IBM VM/CMS->AIX compiler.

Implementation-Defined Attributes

P'OFFSET

Description: Return a value of type LONG_INTEGER that is equal to the address difference between the first storage unit allocated to object P and the base register of the memory page in which object P is located.

This attribute is not supported by the Sun Ada compiler, the Telegen2 Ada compilers Version 3.20, or the Telegen2 Ada compiler Version 3.22 targeted to MIL-STD 1750A.

4.15. TLD Systems, Inc.

Observed in the following compilation systems,
except when noted:

TLD MV/1750A Ada Compiler System Version 1.3.0
Data General MV/32 20000-2, AOS/VS Version 7.60 host
TLD 1750A Instruction Level Simulator, TLD 1750A
Single Program Kernel Version 1.3.0 target
Validation Summary Report, Appendix B (Ada RM Appendix F)

TLD VAX/1750A Ada Compiler System Version 1.3.0
DEC MicroVAX II, MicroVMS Version 4.7 host
TLD 1750A Instruction Level Simulator, TLD 1750A
Single Program Kernel Version 1.3.0 target
Validation Summary Report, Appendix B (Ada RM Appendix F)

TLD HP/1750A Ada Compiler System Version 1.3.0
Hewlett Packard 9000/350, HP-UX Version 6.0 host
HP 64000 with the TASCO MDC281 Emulator, TLD 1750A
Single Program Kernel Version 1.3.0 target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **CONTROLLED**
- **MEMORY_SIZE**
- **OPTIMIZE**
- **SHARED**
- **STORAGE_UNIT**
- **SYSTEM_NAME**

Implementation-Defined Pragmas

EXPORT (language, name [, string_literal])

Description: Directive provides an external name for an object or subprogram. This pragma is identical to the ICC EXPORT pragma.

INCLUDE (file_path_name_string)

Description: Directive specifies that the text in the named file will be included at the place of the pragma. This pragma is identical to ICC INCLUDE pragma.

IF (compile_time_boolean_expression)
ELSIF (compile_time_boolean_expression)
ELSE
END

Description: Directive specifies whether the statements enclosed by the pragmas are to be compiled, depending on the value of the expression. The pragmas are identical to the ICC conditional compilation pragmas.

Implementation-Defined Attributes

No implementation-defined attributes are provided by this vendor.

4.16. Verdix Corp.

Observed in the following compilation systems,
except when noted:

Verdix VADS Ada Version 5.7 Beta-c
SUN-3 UNIX host,
Motorola MC680X0 MVME-133a with MC68881 Floating-Point
Co-processor target
Appendix F

Verdix VADS Ada Version 5.5,
DEC VAX/VMS host,
Motorola MC68020 MVME-133 with MC68881 Floating-Point
Co-processor target
Appendix F

Verdix VADS Ada-010-01125 Version 5.5
DEC MicroVAX II, UNIX 4.2 BSD host
Microbar GPC-68020, bare machine target
Derived host/target:
DEC VAX 8XXX 11/7XX MicroVAX II, UNIX 4.2 BSD, 4.3 BSD,
VAX ULTRIX 1.0, 1.1 host
Microbar GPC-68020 Motorola MVME-133 MC68020, bare machine target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Verdix VADS Ada-010-01125 Version 5.5
DEC MicroVAX II, Micro VMS 4.4 host
Microbar GPC-68020, bare machine target
Derived host/target:
DEC VAX 8XXX 11/7XX Micro VAX II, VAX VMS 4.2, 4.3, 4.4 host
Microbar GPC-68020 Motorola MVME-133 MC68020, bare machine target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Verdix VADS Ada-010-0303 Version 5.5
DEC MicroVAX II, MicroVMS 4.4 host/target
Derived host/target:
DEC VAX 8XXX 11/7XX, VMS 4.2 4.3 4.4 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Verdix VADS Ada-010-5151 Version 5.5
AT&T 3B15, Unix System V release 2.1 host/target
Derived host/target:
AT&T 3B2, Unix System V release 3.1 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

CONVEX C-Series Ada
(C-210,C-130,C-120) CONVEX UNIX Version 6.2 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Encore Verdix Ada Development System Version 5.5
Encore Multimax 320, Umax 4.2 Version R3_1.1 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Elxsi VADS Version 5.5

Elxsi 6400, ENIX 4BSD host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

MIPS/VADS Version 1.30,
MIPS M/1000 UMIPS Version 3.0 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Verdix VADS Ada-010-3131 Version 5.5
Prime EXL, UNIX System V release 3.0 host/target
Derived host/target:
Intel Systems 301 and 320 AT&T 6386, UNIX System V release 3.0
Validation Summary Report, Appendix B (Ada RM Appendix F)

Verdix VADS Ada-010-31315 Version 5.5
Intel System 320, UNIX System V release 3.0 host
Intel iSBC 386/20, bare machine target
Derived host/target:
Intel Systems 301 Prime EXL AT&T 6386, UNIX System V
release 3.0 host
Intel iSBC 386/20, bare machine target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Sequent Ada,
Sequent Symmetry S81, Dynix 3.0.8 host/target
Derived host/target:
Sequent Symmetry S27, Dynix 3.0.8 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Verdix VADS Ada-010-2323 Version 5.5
Sequent Balance 8000, Sequent Dynix Release 2.1 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Silicon Graphics Computer Systems Ada Version 1.0
'ris-4D Entry Systems Workstation, UNIX System V.3 Release 3.5
host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Silicon Graphics Computer Systems Ada Version 1.0
Iris 3000 Series Workstation, UNIX System V Release GL2-W3.6
host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Tolerant Systems, Tolerant Ada Development System Version 2.2
Tolerant Eternity, TX Release 5.3.15 host/target
Validation Summary Report, Appendix B (Ada RM Appendix F)

Language-Defined Pragmas

Appendix F indicates the following language-defined pragmas are not supported:

- **CONTROLLED**
- **OPTIMIZE** *Replaced by multi-level compiler option switch.*
- **SHARED**

The language-defined pragma SUPPRESS has been altered to allow for the specification of arguments such as ALL_CHECKS and EXCEPTION_TABLES, thereby reducing object size by not requiring generation of code and data needed for detecting exceptions; such units will not support exception handling. This modification is only provided for the Verdix VADS 5.7 SUN-3 to MC680X0 implementation.

Implementation-Defined Pragmas

BUILT_IN

Description: Specification unknown. Used by Verdix internal diagnostic services and not available to users.

This pragma not supported by Elxsi VADS compiler.

EXTERNAL_NAME (name, string_literal)

Description: Directive associates an external name with an Ada program name. The effect is to export the Ada identifier. See notes under INTERFACE.

INTERFACE (language_name, subprogram_name [, string_literal])

Description: The last argument provides an external name for the subprogram. Pragma INTERFACE is thereby supported in a non-conforming manner, yet retains the functionality intended.

IMPLICIT_CODE ([ON] | [OFF])

Description: Directive specifies that implicit code generated by the compiler (code generated to support subprogram entry and exit calling conventions) is allowed (ON) or disallowed (OFF). The default is ON. Use is limited to parameterless calls of inlined machine-code routines. This pragma is similar to the Harris IMPLICIT_CODE pragma.

OPTIMIZE_CODE ([OFF | ON])

Description: Directive specifies whether the compiler should attempt to optimize machine code insertions. When OFF is specified, the compiler will generate the code as specified.

This pragma not supported by Elxsi VADS compiler.

INLINE_ONLY

Description: Directive specifies that the subprogram must always be compiled inline. The pragma obeys the same rules as the language-defined pragma INLINE (which the vendor supports) except that a callable version of the subprogram is not generated, as a space saving optimization. The pragma INLINE is supported to a nested depth of eight when the body is available for inline expansion. One can surmise (since it is not stated) that the non-availability of a body will result in abnormal termination of the compilation at that point.

INTERFACE_NAME (name, string_literal)

Description: Directive provides an external name for subprograms or objects. The object must not be one of the following: a loop variable, a constant, an initialized variable, an array, or a record.

This pragma is only supported by the Verdix VADS 5.7 SUN-3 to MC680X0 compiler.

INTERFACE_OBJECT (variable_name, string_literal)

Description: Directive provides an external name for a variable. The object must not be one of the following: a loop variable, a constant, an initialized variable, an array, or a record.

This pragma is not supported by the Verdix VADS 5.7 SUN-3 to MC680X0 compiler.

NO_IMAGE

Description: Directive suppresses generation of the table used for the IMAGE attribute of enumeration types. The pragma reduces the amount of runtime storage neeaed.

NON_REENTRANT (subprogram_name)

Description: Directive specifies that the named subprogram, which must be a library subprogram or a subprogram declared in the specification or body part of a package, will not be called by different tasks. The effect is to allow the compiler to perform certain optimizations that would not normally be attempted (such as automatic inlining).

This pragma is only supported by the Verdix VADS 5.7 SUN-3 to MC680X0 implementation.

NO_RECURRENCE

Description: Directive forces loop vectorization The existence of a vector recurrence in loop will cause incorrect code generation if the pragma is in effect.

This pragma is only supported by CONVEX Ada compiler for CONVEX C-Series targets.

NO_SIDE_EFFECTS

Description: Directive specifies that the subprogram identified by the compilation unit in which the pragma is given does not modify the value of objects outside the subprogram. This pragma is a global optimization feature.

This pragma is only supported by the CONVEX Ada compiler for CONVEX C-Series targets.

NOT_ELABORATED

Description: Directive specifies that elaboration code need not be generated. A compiler warning is issued if elaboration code is needed. The pragma only applies to 1) packages that are part of the compiler runtime system, 2) a configuration package, or 3) an Ada package that is referenced from another language. The pragma may only be placed in the specification part of a library package unit.

This pragma is only supported by the Verdix VADS 5.7 SUN-3 to MC680X0 implementation.

PASSIVE

PASSIVE(SEMAPHORE)

PASSIVE(INTERRUPT, NUMBER)

Description: Use of the pragma within a task or task type allows more efficient entry calls to be generated. In particular, such calls check a semaphore and then call an accept statement as if it were a procedure. If the semaphore is unlocked, the cost of such an entry call is essentially the cost of a procedure call, since there is no context switching overhead.

The form with the SEMAPHORE argument is equivalent to the parameterless form. The INTERRUPT, NUMBER form of the pragma causes the passive task to be mapped to an interrupt vector for a hardware entry call. When an interrupt entry is called, further interrupts are turned off (using NUMBER as a bit mask) until the accept statement has completed its execution. Certain statements within a task body may prevent the optimization from occurring. In these cases a compile-time warning will be issued and the exception TASKING_ERROR will be raised at execution time.

The passive task or task type must be declared immediately within a package specification or body. There are no restrictions on entry parameters or the declaration of local subprograms within the body of a PASSIVE task. Passive tasks may not have:

- entry families
- nested accepts
- declare blocks
- delay statements outside an accept block (disallowing delay alternatives in a selective_wait)
- entry calls outside of an accept block (including conditional and timed)
- pragma PRIORITY
- entry count attribute P'COUNT
- task type for a PASSIVE interrupt entry task
- interrupt entries for a PASSIVE semaphore task

This pragma is only supported by the Verdix VADS 5.7 SUN-3 to MC680X0 implementation.

SCALAR

Description: Directive specifies that a loop shall not be vectorized. Nested inner loops remain eligible for vectorization.

This pragma is supported only by CONVEX Ada compiler for CONVEX C-Series targets.

SHARE_CODE (generic_unit_name [, TRUE | FALSE])

Description: Directive specifies the sharing of generic bodies between multiple instantiations of the same generic library unit. The pragma is only effective when two or more instantiations have identical types as actual parameters. The pragma SHARE_BODY has the same effect. The named generic unit may be a generic package or a generic instantiation. If the second argument is TRUE, code is shared between the instantiations (or for all instantiations when a generic unit name is provided).

VECTOR_UNIT

Description: Directive specifies that a library unit is eligible for vectorization. All containing loops are analyzed for this optimization and those which have no vector recurrence dependencies are converted to a vectorized operation. Further restrictions apply to the multiple use of pragmas SCALAR and VECTOR_UNIT within a library unit.

This pragma is supported only by the CONVEX Ada compiler for CONVEX C-Series targets.

Implementation-Defined Attributes

P'REF

Description: If P is a constant, variable, procedure, function, or label, the attribute returns a value of the type MACHINE_CODE.OPERAND. The pragma is used to designate an operand within a code statement. Another form of the pragma is as SYSTEM.ADDRESS'REF(N) where an integer expression N, of type *universal_integer*, is converted and returned as an address. For the CONVEX Ada compiler, this attribute is of type OPERAND as defined in system package MACHINE_CODE and is restricted to use with machine code insertions.

References

[ACS 89] Weiderman, Nelson H.
Ada Adoption Handbook: Compiler Evaluation and Selection
Carnegie-Mellon University, Software Engineering Institute
Technical Report CMU/SEI-89-TR-13
ESD-TR-89-21
Version 1.0
March 1989

[AIC 89] Ada Joint Program Office (AJPO)
Ada Information Clearinghouse (AdaIC) Newsletter
The Pentagon, Washington, D.C.
Volume VII, Number 2
July 1989

[AVO 88] Ada Validation Organization (AVO)
Institute for Defense Analyses
Ada Validation Facility (AVF)
Ada Compiler Validation Summary Report in reference to
Ada Compiler Validation Procedures and Guidelines
Ada Joint Program Office (AJPO), 1 January 1987

[RM 83] *Reference Manual for the Ada Programming Language*, 1983
ANSI/MIL-STD-1815A-1983

Index